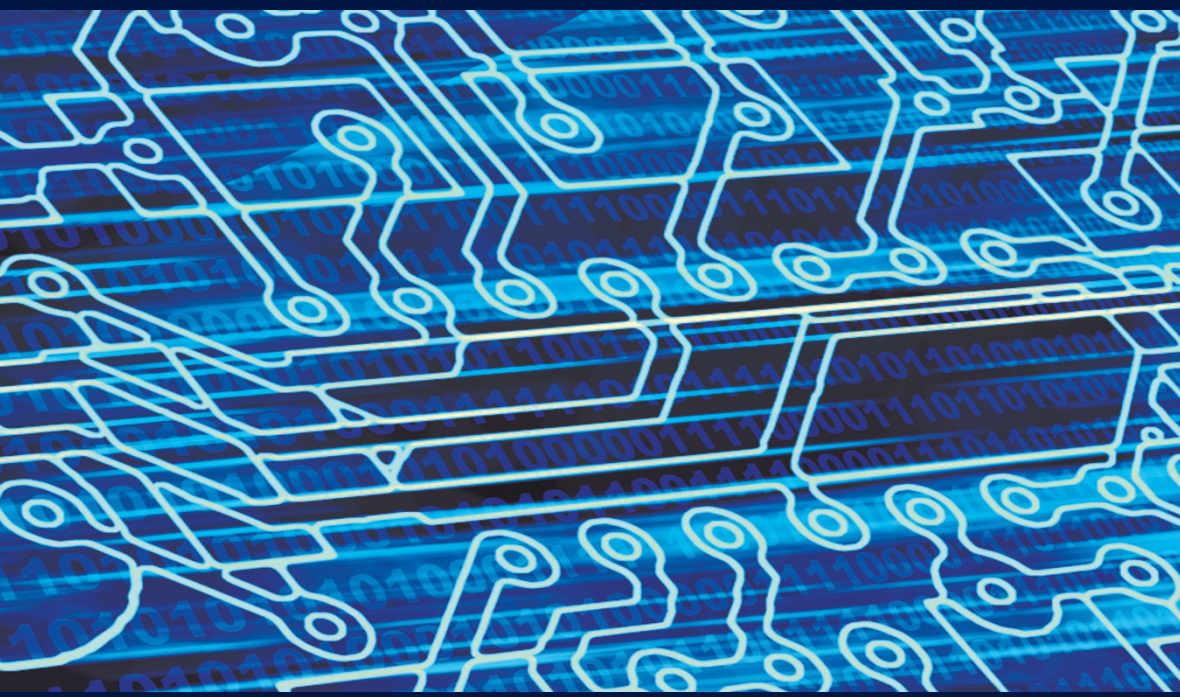
ELECTRONICS ENGINEERING SERIES

Digital Electronics 1

Combinational Logic Circuits

Tertulien Ndjountche



iSTE WILEY

VISIT…

Digital Electronics 1

Series Editor
Robert Baptist

Digital Electronics 1

Combinational Logic Circuits

Tertulien Ndjountche

ISTE

WILEY

First published 2016 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms and licenses issued by the CLA. Enquiries concerning reproduction outside these terms should be sent to the publishers at the undermentioned address:

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

© ISTE Ltd 2016
The rights of Tertulien Ndjountche to be identified as the author of this work have been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

Library of Congress Control Number: 2016939642

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-84821-984-7

Contents

Preface ix

Chapter 1. Number Systems 1

1.1. Introduction 1
1.2. Decimal numbers 1
1.3. Binary numbers 2
1.4. Octal numbers 4
1.5. Hexadecimal numeration 5
1.6. Representation in a radix B 6
1.7. Binary-coded decimal numbers 7
1.8. Representations of signed integers 8
1.8.1. Sign-magnitude representation 9
1.8.2. Two's complement representation 10
1.8.3. Excess-E representation 12
1.9. Representation of the fractional part of a number 13
1.10. Arithmetic operations on binary numbers 16
1.10.1. Addition 16
1.10.2. Subtraction 17
1.10.3. Multiplication 18
1.10.4. Division 19
1.11. Representation of real numbers 20
1.11.1. Fixed-point representation 20
1.11.2. Floating-point representation 22
1.12. Data representation 28
1.12.1. Gray code 28
1.12.2. p-out-of-n code 29
1.12.3. ASCII code 31
1.12.4. Other codes 31
1.13. Codes to protect against errors 31

1.13.1. Parity bit . . . 31
1.13.2. Error correcting codes . . . 33
1.14. Exercises . . . 36
1.15. Solutions . . . 38

Chapter 2. Logic Gates . . . 49

2.1. Introduction . . . 49
2.2. Logic gates . . . 50
2.2.1. NOT gate . . . 51
2.2.2. AND gate . . . 51
2.2.3. OR gate . . . 52
2.2.4. XOR gate . . . 52
2.2.5. Complementary logic gates . . . 53
2.3. Three-state buffer . . . 54
2.4. Logic function . . . 54
2.5. The correspondence between a truth table and a logic function . . . 55
2.6. Boolean algebra . . . 57
2.6.1. Boolean algebra theorems . . . 59
2.6.2. Karnaugh maps . . . 65
2.6.3. Simplification of logic functions with multiple outputs . . . 73
2.6.4. Factorization of logic functions . . . 74
2.7. Multi-level logic circuit implementation . . . 76
2.7.1. Examples . . . 77
2.7.2. NAND gate logic circuit . . . 78
2.7.3. NOR gate based logic circuit . . . 80
2.7.4. Representation based on XOR and AND operators . . . 82
2.8. Practical considerations . . . 89
2.8.1. Timing diagram for a logic circuit . . . 90
2.8.2. Static hazard . . . 90
2.8.3. Dynamic hazard . . . 92
2.9. Demonstration of some Boolean algebra identities . . . 93
2.10. Exercises . . . 97
2.11. Solutions . . . 101

Chapter 3. Function Blocks of Combinational Logic . . . 115

3.1. Introduction . . . 115
3.2. Multiplexer . . . 115
3.3. Demultiplexer and decoder . . . 121
3.4. Implementation of logic functions using multiplexers or decoders . . . 127
3.4.1. Multiplexer . . . 127
3.4.2. Decoder . . . 129
3.5. Encoders . . . 130
3.5.1. 4:2 encoder . . . 131

3.5.2. 8:3 encoder . . . 134
3.5.3. Priority encoder . . . 136
3.6. Transcoders . . . 143
3.6.1. Binary code and Gray code . . . 143
3.6.2. BCD and excess-3 code . . . 149
3.7. Parity check generator . . . 155
3.8. Barrel shifter . . . 160
3.9. Exercises . . . 165
3.10. Solutions . . . 173

Chapter 4. Systematic Methods for the Simplification of Logic Functions . . . 203

4.1. Introduction . . . 203
4.2. Definitions and reminders . . . 203
4.2.1. Definitions . . . 204
4.2.2. Minimization principle of a logic function . . . 204
4.3. Karnaugh maps . . . 205
4.3.1. Function of five variables . . . 205
4.3.2. Function of six variables . . . 207
4.3.3. Karnaugh map with entered variable . . . 208
4.3.4. Applications . . . 215
4.3.5. Representation based on the XOR and AND operators . . . 220
4.4. Systematic methods for simplification . . . 220
4.4.1. Determination of prime implicants . . . 221
4.4.2. Finding the constitutive terms of a minimal expression . . . 224
4.4.3. Quine–McCluskey technique: simplification of incompletely defined functions . . . 235
4.4.4. Simplification of functions with multiple outputs . . . 235
4.5. Exercises . . . 241
4.6. Solutions . . . 243

Bibliography . . . 257

Index . . . 259

Preface

The omnipresence of electronic devices in everyday life is accompanied by the decreasing size and the ever-increasing complexity of digital circuits. This comprehensive and easy-to-understand work deals with the basic principles of digital electronics and allows the reader to grasp the subtleties of digital circuits, from logic gates to finite-state machines. It presents all the aspects related to combinational logic and sequential logic. It introduces techniques for simply and concisely establishing logic equations as well as methods for the analysis and design of digital circuits. Emphasis has been especially laid on design approaches that can be used to ensure a reliable operation of finite-state machines. Various programmable logic circuit structures and their applications have also been presented. Each chapter is completed by practical examples and well-designed exercises that are accompanied by worked solutions.

This book discusses all the different aspects of digital electronics, using a descriptive approach combined with a gradual, detailed and comprehensive presentation of basic concepts. The principles of combinational and sequential logic are presented, as well as the underlying techniques to the analysis and design of digital circuits. The analysis and design of digital circuits with increasing complexity is facilitated by the use of abstractions at the circuit and architecture levels. There are three volumes in this series devoted to the following subjects:

1) combinational logic circuits;

2) sequential and arithmetic logic circuits;

3) finite-state machines.

A progressive approach has been chosen and the chapters are relatively independent of each other. To help master the subject matter and put into practice the different concepts and techniques, the books are complemented by a selection of exercises and solutions.

1. Summary

Volume 1 deals with combinational logic circuits. Logic gates are basic components in digital circuits. They implement Boolean logic functions and operations that are applied to binary-coded data. Combinational logic is used only for logic functions and operations whose outputs depend solely on the inputs. This first volume contains the following four chapters:

1) Number Systems;

2) Logic Gates;

3) Function Blocks of Combinational Logic;

4) Systematic Methods for the Simplification of Logic Functions.

2. The reader

This book is an indispensable tool for all engineering students in bachelors or masters course who wish to acquire detailed and practical knowledge of digital electronics. It is detailed enough to serve as a reference for electronic, automation and computer engineers.

Tertulien NDJOUNTCHE
April 2016

1

Number Systems

1.1. Introduction

Digital systems are used to process data and to perform calculations in most instrumentation, monitoring and communication devices. As physical quantities and signals can only take discrete values in a digital system, the interpretation of real-world information requires the use of interface circuits such as data converters.

In general, numbers may be represented in different numeration systems. The decimal system is commonly used in routine transactions while the binary system is the basis for digital electronics. Every number (or numeration) system is defined by a base (or *radix*), which is a collection of distinct symbols. The representation of a number in a numeration system may be considered as a change in base. In a positional number system, a value of a number depends on the place occupied by each of its digits in the representation.

1.2. Decimal numbers

The decimal number system uses the following 10 numbers or symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The radix is thus 10.

EXAMPLE 1.1.– Decompose the numbers 734 and 12345 into powers of 10.

The decomposition of the number 734 takes the form:

$$\begin{aligned} 734 &= (7 \times 10^2) + (3 \times 10^1) + (4 \times 10^0) \\ &= 734_{10} \end{aligned}$$

Digital Electronics 1: *Combinational Logic Circuits*, First Edition. Tertulien Ndjountche.
© ISTE Ltd 2016. Published by ISTE Ltd and John Wiley & Sons, Inc.

For the number 12345, we have:

$$\begin{aligned} 12\ 345 &= (1 \times 10^4) + (2 \times 10^3) + (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0) \\ &= 12\ 345_{10} \end{aligned}$$

Depending on its position, each number is multiplied by the appropriate power of 10. The right-most digit represents the unit digit.

1.3. Binary numbers

Binary number system is based on two-level logic, conventionally noted as 0 (low level) and 1 (high level). It is a system with a radix of two.

EXAMPLE 1.2.– Convert the decimal numbers 13 and 125 into binary numbers.

The decomposition of the number 13 in powers of 2 is written as:

$$\begin{aligned} 13_{10} &= (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ &= 1101_2 \end{aligned}$$

For the number 125, we have:

$$\begin{aligned} 125_{10} &= (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) \\ &\quad +(1 \times 2^0) = 1111101_2 \end{aligned}$$

The binary code that is then obtained for a positive number is called a natural binary code.

The coefficients or numbers (0 or 1) used in the binary representation of a number are called bits.

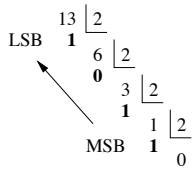
The right-most bit is called the *least significant bit* (LSB), while the left-most bit is called the *most significant bit* (MSB).

In practice, the conversion of a decimal number to a binary number can be carried out by reading, from last to first, the remainders of a series of integer divisions as illustrated by Figure 1.1.

The arithmetic and logic unit of a microprocessor manipulates binary numbers or *words* with a fixed number of bits.

$13 \div 2 = 6$ Remainder 1 (LSB)
$6 \div 2 = 3$ Remainder 0
$3 \div 2 = 1$ Remainder 1
$1 \div 2 = 0$ Remainder 1 (MSB)

$13_{10} = 1101_2$



$125 \div 2 = 62$ Remainder 1 (LSB)
$62 \div 2 = 31$ Remainder 0
$31 \div 2 = 15$ Remainder 1
$15 \div 2 = 7$ Remainder 1
$7 \div 2 = 3$ Remainder 1
$3 \div 2 = 1$ Remainder 1
$1 \div 2 = 0$ Remainder 1 (MSB)

$125_{10} = 1111101_2$

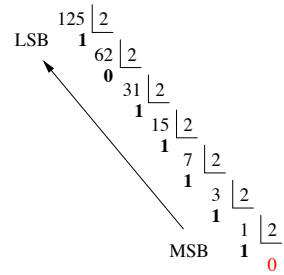


Figure 1.1. *Decimal-binary conversion using successive division methods*

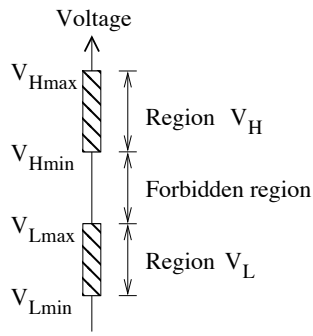


Figure 1.2. *Representation of logic voltage levels*

A *byte* is an 8-bit word.

In practice, the bits 0 and 1 are represented by voltage or current levels. Figure 1.2 shows the representation of logic voltage levels. The two regions V_H and V_B are separated by a forbidden region where the logical level is undefined.

Logical states may be assigned to regions based on positive logic or negative logic. In the case of positive logic, the region V_H corresponds to 1 (or the high level), and

the region V_B corresponds to 0 (or the low level); and in the case of negative logic, the region V_H corresponds to 0 (or low level), and the region V_B corresponds to 1 (or high level).

1.4. Octal numbers

The octal number system or a representation with radix eight consists of the following symbols: 0, 1, 2, 3, 4, 5, 6, 7.

EXAMPLE 1.3.– Convert the decimal numbers 250 and 777 to octal numbers.

In radix 8 representation, the number 250 takes the form:

$$\begin{aligned} 250_{10} &= (3 \times 8^2) + (7 \times 8^1) + (2 \times 8^0) \\ &= 372_8 \end{aligned}$$

In the case of the number 777, we have:

$$\begin{aligned} 777_{10} &= (1 \times 8^3) + (4 \times 8^2) + (1 \times 8^1) + (1 \times 8^0) \\ &= 1\,411_8 \end{aligned}$$

The right-most digit is called the *least significant digit* (LSD), while the left-most digit is called the *most significant digit* (MSD).

A practical approach to converting a decimal number to an octal number consists of carrying out a series of integer divisions as illustrated in Figure 1.3.

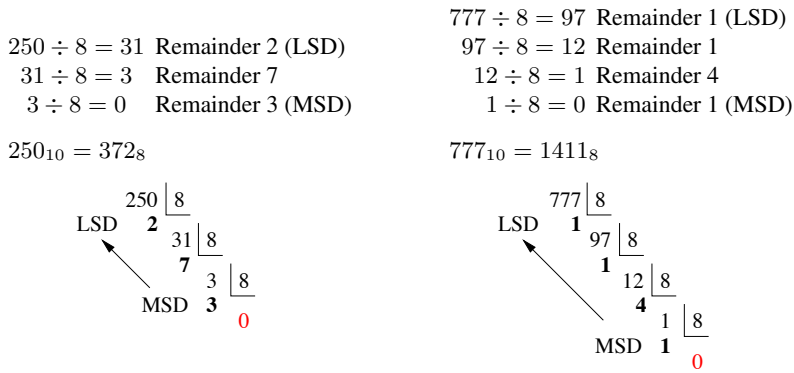


Figure 1.3. *Decimal-octal conversion using the successive division method*

Octal numeration may be deduced from binary numeration by grouping, beginning from the right, consecutive bits in triplets or, conversely, by replacing each octal number by its three corresponding bits.

EXAMPLE 1.4.– Determine the radix 8 representation for the decimal numbers 85 and 129.

Radix 8 representations are obtained by replacing each group of three bits by the equivalent octal number. We can therefore write:

$$85_{10} = 1010101_2 = \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{101}_{5} = 125_8$$

Similarly,

$$129_{10} = 10000001_2 = \underbrace{010}_{2}\ \underbrace{000}_{0}\ \underbrace{001}_{1} = 201_8$$

1.5. Hexadecimal numeration

The hexadecimal number system or a representation with a radix 16 consists of the following symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

EXAMPLE 1.5.– Convert the decimal numbers 291 and 1000 to hexadecimal.

The number 291 is represented in radix 16 by:

$$\begin{aligned} 291_{10} &= (1 \times 16^2) + (2 \times 16^1) + (3 \times 16^0) \\ &= 123_{16} \end{aligned}$$

For the number 1000, we obtain:

$$\begin{aligned} 1\,000_{10} &= (3 \times 16^2) + (14 \times 16^1) + (8 \times 16^0) \\ &= 3E8_{16} \end{aligned}$$

In practice, a series of integer divisions makes it possible to convert a decimal number to a hexadecimal number. The different remainders constitute the results of the conversion, beginning with the last, which is the MSD, to the first, which represents the LSD. We thus have:

$291 \div 16 = 18$ Remainder 3 (LSD)
$18 \div 16 = 1$ Remainder 2
$1 \div 16 = 0$ Remainder 1 (MSD)

$291_{10} = 123_{16}$

$1000 \div 16 = 62$ Remainder 8 (LSD)
$62 \div 16 = 3$ Remainder 14
$3 \div 16 = 0$ Remainder 3 (MSD)

$1000_{10} = 3E8_{16}$

Figure 1.4. *Decimal-hexadecimal conversion using the successive division method*

We can also proceed as demonstrated in Figure 1.4, the result of each conversion being made up of the successive remainders of the divisions.

Binary to hexadecimal conversion is done by grouping the bits representing the binary four by four and beginning from the right, conversely, replacing each hexadecimal digit by its four corresponding bits.

EXAMPLE 1.6.– Convert the decimal numbers 31 and $2,988$ into hexadecimal.

To obtain the equivalent hexadecimal from the binary representation, each group of four bits is replaced by the corresponding hexadecimal digit. We therefore have:

$$31_{10} = 11111_2 = \underbrace{0001}_{1}\ \underbrace{1111}_{15=F} = 1F_{16}$$

Similarly,

$$2\,988_{10} = 101110101100_2 = \underbrace{1011}_{11=B}\ \underbrace{1010}_{10=A}\ \underbrace{1100}_{12=C} = BAC_{16}$$

It is generally more convenient to represent the value of an octet using two hexadecimal digits as it is more compact.

1.6. Representation in a radix B

In general, in radix B representation, a decimal number N may be decomposed as follows:

$$N_{10} = b_{n-1}B^{n-1} + \cdots + b_2B^2 + b_1B^1 + b_0B^0 \qquad [1.1]$$

$$= \sum_{i=0}^{n-1} b_iB^i \qquad [1.2]$$

where $B \geq 2$. Thus, the decimal number N is represented in radix B with n digits, $b_{n-1} \cdots b_2b_1b_0$.

Using n digits in a radix B numeration, we can code the decimal numbers from 0 to $B^n - 1$.

For an integer represented by n digits with a radix B, the formulas for conversion are as follows:

$$\begin{aligned}(b_{n-1}b_{n-2}\cdots b_2b_1b_0)_B &= \sum_{i=0}^{n-1} b_iB^i \\ &= b_{n-1}B^{n-1} + b_{n-2}B^{n-2} + \cdots + b_2B^2 + b_1B^1 + b_0B^0 \\ &= b_0 + B(b_1 + B(b_2 + B(\cdots + B(b_{n-2} + Bb_{n-1})\cdots))) \\ &= N_{10}\end{aligned} \qquad [1.3]$$

EXAMPLE 1.7.– Convert the binary number 110101_2, the octal number 5671_8 and the hexadecimal number $5CAD_{16}$ to decimal.

In decimal form, the number 110101_2 is written as:

$$\begin{aligned}110111_2 &= \mathbf{1}\times 2^5 + \mathbf{1}\times 2^4 + \mathbf{0}\times 2^3 + \mathbf{1}\times 2^2 + \mathbf{1}\times 2^1 + \mathbf{1}\times 2^0 \\ &= \mathbf{1} + 2(\mathbf{1} + 2(\mathbf{1} + 2(\mathbf{0} + 2(\mathbf{1} + 2\times\mathbf{1})))) \\ &= 55_{10}\end{aligned}$$

For the number 5671_8, we get:

$$\begin{aligned}5671_8 &= \mathbf{5}\times 8^3 + \mathbf{6}\times 8^2 + \mathbf{7}\times 8^1 + \mathbf{1}\times 8^0 \\ &= \mathbf{1} + 8(\mathbf{7} + 8(\mathbf{6} + 8\times\mathbf{5})) \\ &= 3001_{10}\end{aligned}$$

The conversion of the number $5CAD_{16}$ to decimal is effected by:

$$\begin{aligned}5CAD_{16} &= \mathbf{5}\times 16^3 + \mathbf{12}\times 16^2 + \mathbf{10}\times 16^1 + \mathbf{13}\times 16^0 \\ &= \mathbf{13} + 16(\mathbf{10} + 16(\mathbf{12} + 16\times\mathbf{5})) \\ &= 23725_{10}\end{aligned}$$

1.7. Binary-coded decimal numbers

To represent a 8421-type *binary-coded decimal* (BCD) number, each digit must be replaced by its equivalent 4-bit binary.

EXAMPLE 1.8.– Give the BCD representation for the decimal numbers 90 and 873.

The BCD representation of the number 90 is written as follows:

$$90_{10} = 1001\ 0000_{BCD}$$

For the number 873, we have:

$$873_{10} = 1000\ 0111\ 0011_{BCD}$$

Table 1.1 gives the hexadecimal, octal, binary and BCD representations of numbers from 0 to 15.

Decimal number	Representation			
	Hexadecimal	Octal	Binary	BCD
0	0	0	0000	0000
1	1	1	0001	0001
2	2	2	0010	0010
3	3	3	0011	0011
4	4	4	0100	0100
5	5	5	0101	0101
6	6	6	0110	0110
7	7	7	0111	0111
8	8	10	1000	1000
9	9	11	1001	1001
10	A	12	1010	0001 0000
11	B	13	1011	0001 0001
12	C	14	1100	0001 0010
13	D	15	1101	0001 0011
14	E	16	1110	0001 0100
15	F	17	1111	0001 0101

Table 1.1. *Conversion tables for 0 numbers to 15*

It must be noted that with n bits, we can represent the decimal numbers between 0 and $10^{n/4} - 1$. In addition to the 8421 BCD code, there are other types of BCD codes.

1.8. Representations of signed integers

Several approaches may be adopted to represent signed integers in digital systems: the *sign-magnitude (SM) representation*, *two's complement (2C) representation*, and *excess-E (XSE) representation*. Each of these approaches assumes the use of a format (or number of bits) fixed beforehand.

1.8.1. *Sign-magnitude representation*

The simplest approach allowing for the representation of a signed integer consists of reserving the MSB for the number sign and the remaining bits for the number magnitude. If the sign bit is set to 0, the number is positive, and if the sign bit is set to 1, the number is negative.

EXAMPLE 1.9.– Using 8 bits, determine the sign-magnitude representation for each of the decimal numbers 55, −60, and 0.

We have:

$$55_{10} = 00110111_2 \quad \text{and} \quad 55_{10} = 00110111_{SM}$$

$$60_{10} = 00111100_2 \quad \text{and} \quad -60_{10} = 10111100_{SM}$$

In the case of 0, two representations are possible:

$$+0_{10} = 00000000_{SM} \quad \text{and} \quad -0_{10} = 10000000_{SM}$$

The value of a decimal number N having an sign-magnitude representation of the form $b_{n-1}b_{n-2}\cdots b_0$ is given by:

$$N_{10} = (-1)^{b_{n-1}} \sum_{i=0}^{n-2} b_i 2^i \qquad [1.4]$$

or

$$N_{10} = (1 - 2b_{n-1}) \sum_{i=0}^{n-2} b_i 2^i \qquad [1.5]$$

In this way, it is possible to represent the numbers in the range from $-(2^{n-1}-1)$ to $2^{n-1}-1$, using n bits.

However, the sign-magnitude representation presents two problems. The first is linked to the two representations, $+0$ and -0, of the number 0. The second problem arises from the fact that this representation is not appropriate for addition operations, especially when one of the numbers is negative. The two's complement representation allows us to remedy these two problems.

1.8.2. *Two's complement representation*

Two's complement representation of a number with n bits actually corresponds to the complement with respect to 2^n and is defined as the difference between 2^n and this number in absolute value.

EXAMPLE 1.10.– Determine the 8-bit two's complement representation of the numbers 90 and -120.

As the number 90 is positive, the two's complement representation is identical to the natural binary representation:

$$90_{10} = 01011010_2 = 01011010_{2C}$$

The number -120 is negative and the two's complement representation is obtained as follows:

$$2^8 - 120 = 136 \text{ and } 136_{10} = 10001000_2 \text{ from which } -120_{10} = 10001000_{2C}$$

Similarly, the two's complement representation of a number may be obtained by taking the one's complement and then adding 1 (ignoring the overflow), because the sum of a number and its one's complement is equal to a number having all bits at 1 (or high logic level).

NOTE 1.1.– Assuming that the binary representation using n bits, of a positive number N takes the form, $b_{n-1}b_{n-2}\cdots b_1b_0$, the two's complement representation of $-N$ may be written as follows:

$$\begin{aligned} 2^n - N &= (2^n - 1) - N + 1 \\ &= \underbrace{111\cdots11}_{n \text{ bits}}{}_2 - b_{n-1}b_{n-2}\cdots b_1b_0 + 1 \end{aligned} \quad [1.6]$$

where

$$2^n - 1 = \underbrace{111\cdots11}_{n \text{ bits}}{}_2$$

and the subtraction

$$\underbrace{111\cdots11}_{n \text{ bits}}{}_2 - b_{n-1}b_{n-2}\cdots b_1b_0 \quad [1.7]$$

allows for the inversion of each bit of N or for obtaining the one's complement of N.

EXAMPLE 1.11.– The application of the above-cited method to determine the two's complement of the decimal number -120 using 8 bits translates to:

	01111000	Binary representation of the decimal number 120
	10000111	One's complement obtained by inverting each bit
+	1	Addition of 1
	10001000	Two's complement

and

$$-120_{10} = 10001000_{2C}$$

The value of a decimal number N with two's complement representation taking the form, $b_{n-1}b_{n-2}\cdots b_0$, is given by:

$$N_{10} = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \qquad [1.8]$$

Using n bits, we can represent the numbers in the range from -2^{n-1} to $2^{n-1}-1$.

In the case of 8-bit two's complement representation, the highest positive value is:

$$2^{8-1} - 1 = 127_{10} = 01111111_{2C}$$

and the smallest negative value is:

$$-2^{8-1} = -128_{10} = 10000000_{2C}$$

NOTE 1.2.– To obtain two's complement representation from the binary representation of the corresponding positive number, we must:

– identify the first 1 bit beginning from the right;

– take the one's complement for each bit located before the identified bit.

Let us determine the 8-bit two's complement representation for each of the numbers -10_{10} and -119_{10}.

Applying the procedure given in the previous note, as illustrated in Figure 1.5, two's complement representations are given by:

$$-10_{10} = 11110110_{2C} \text{ and } -119_{10} = 01110111_{2C}$$

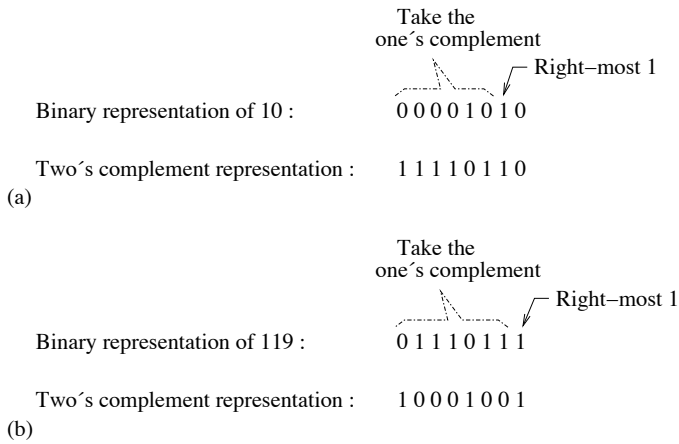

Take the
one´s complement
Right–most 1
Binary representation of 10 :
0 0 0 0 1 0 1 0
Two´s complement representation :
1 1 1 1 0 1 1 0
(a)
Take the
one´s complement
Right–most 1
Binary representation of 119 :
0 1 1 1 0 1 1 1
Two´s complement representation :
1 0 0 0 1 0 0 1
(b)

Figure 1.5. *Obtaining a two's complement from the binary representation: a) -10_{10} and b) -119_{10}*

1.8.3. *Excess-E representation*

Some systems use the excess-E representation in order to be able to represent negative numbers.

In excess-E representations, a number with n bits, whose unsigned value is N, where $0 \leq N \leq N_{\max} = 2^n - 1$, represents the signed integer $N - E$, where E is the offset of the code. We can, thus, represent signed numbers in the range from $-E$ to $N_{\max} - E$. The value of the offset is, most often, of the form $E = 2^{n-1}$ or $E = 2^{n-1} - 1$.

1.8.3.1. *Case where $E = 2^{n-1}$*

Using an excess-2^{n-1} code, any number N in the range from -2^{n-1} to $2^{n-1} - 1$ is represented by the n-bit binary number, $N + 2^{n-1}$, which is always positive and less than 2^n.

EXAMPLE 1.12.– Assuming that $E = 2^{n-1}$, where $n = 4$, determine the excess-E' representation of the decimal numbers 3 and -6.

The excess-8 code for the number 3 is obtained by determining the binary code for the result of the operation $3 + 8 = 11$, that is: $11_2 = 1011_2$. Thus:

$$3_{10} = 1011_{XS8}$$

For the number -6, we have $-6 + 8 = 2$ and $2_{10} = 0010_2$. As a result:

$$-6_{10} = 0010_{XS8}$$

The excess-2^{n-1} code corresponds to a two's complement representation where the sign bit is complemented (1 is replaced by 0 and vice versa).

1.8.3.2. *Case where* $E = 2^{n-1} - 1$

With an excess-$2^{n-1} - 1$ code, we can represent the numbers N in the range from $-(2^{n-1} - 1)$ to 2^{n-1}.

A code similar to the excess-$2^{n-1} - 1$ code is adopted in the standard IEEE-754 used for the representation of the exponents of floating-point numbers.

EXAMPLE 1.13.– Represent the decimal numbers 27 and -43 using the excess-$2^{n-1} - 1$ code, where $n = 8$.

When $n = 8$, the value of the offset is $E = 2^{8-1} - 1 = 2^7 - 1 = 127$.

The excess-127 code for the number 27 is obtained by adding 127 to 27, and then converting the result to binary. That is:

$$27 + 127 = 154 \quad 154_{10} = 10011010_2 \quad \text{and} \quad 27_{10} = 10011010_{XS127}$$

For the excess-127 of the number -43, we have:

$$-43 + 127 = 84 \quad 84_{10} = 01010100_2 \quad \text{and} \quad -43_{10} = 01010100_{XS127}$$

Table 1.2 gives the representations of unsigned and signed 3-bit integers. It must be noted that in sign-magnitude representations, the decimal number 0 has two codes, $+0_{10} = 000_{SM}$ and $-0_{10} = 100_{SM}$. Using 3 bits, the two's complement representation allows for the coding of numbers from 3 to -4, while for the excess-3 representation, the numbers are in the range from 4 to -3.

1.9. Representation of the fractional part of a number

A number is usually made up of an integer part and a fractional part, whose value is lower than 1. The fractional part of a number may be expressed as the sum of the negative powers of the radix of the numeration system.

The number 0.59375 is written in decimal representation as follows:

$$0.59375_{10} = (5 \times 10^{-1}) + (9 \times 10^{-2}) + (3 \times 10^{-3}) + (7 \times 10^{-4}) + (5 \times 10^{-5})$$

Decimal number	Representation			
	Binary	SM	2C	XS3
7	111			
6	110			
5	101			
4	100			111
3	011	011	011	110
2	010	010	010	101
1	001	001	001	100
0	000	000 100	000	011
−1		101	111	010
−2		110	110	001
−3		111	101	000
−4			100	

Table 1.2. *Representations of unsigned and signed 3-bit integers*

It can be converted into binary, octal and hexadecimal, as given below:

$$\begin{aligned} 0.59375_{10} &= (1 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4}) + (1 \times 2^{-5}) \\ &= 0.10011_2 \\ &= 0.\underbrace{100}_{4}\ \underbrace{110}_{6} = 0.46_8 \\ &= 0.\underbrace{1001}_{9}\ \underbrace{1000}_{8} = 0.98_{16} \end{aligned}$$

The practical method to convert the fractional part of a number consists of carrying out a series of multiplications while extracting the integer part each time.

The different operations needed to convert the decimal number 0.59375 are shown in Figure 1.6:

– conversion to binary:

$0.59375 \times 2 = 1.1875$ Integer part 1 (MSB)
$0.1875 \times 2 = 0.375$ Integer part 0
$0.375 \times 2 = 0.75$ Integer part 0
$0.75 \times 2 = 1.5$ Integer part 1
$0.5 \times 2 = 1.0$ Integer part 1 (LSB)

$0.59375_{10} = 0.10011_2$

– conversion to octal:

$0.59375 \times 8 = 4.75$ Integer part 4 (MSD)
$0.75 \times 8 = 6.0$ Integer part 6 (LSD)

$0.59375_{10} = 0.46_8$

– conversion to hexadecimal:

$0.59375 \times 16 = 9.5$ Integer part 9 (MSD)
$0.50 \times 16 = 8.0$ Integer part 8 (LSD)

$0.59375_{10} = 0.98_{16}$

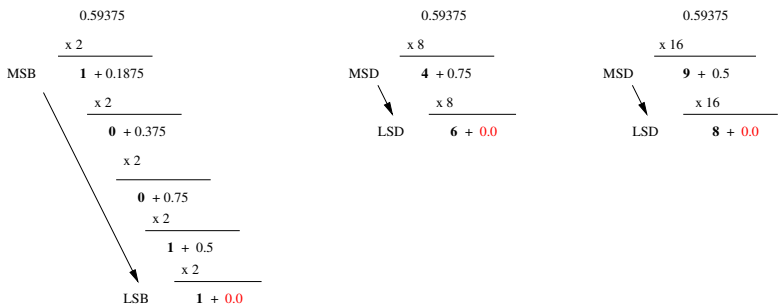


Figure 1.6. *Conversion of the decimal number* 0.59375 *using the successive multiplication method*

NOTE 1.3.– Converting certain fractional numbers produces an infinite sequence of bits.

Convert the decimal number 0.45 to binary. Successively multiplying by 2 and retaining the integer part of the result each time, we obtain:

$0.45 \times 2 = 0.9$ Integer part 0 (MSB)
$0.9 \times 2 = 1.8$ Integer part 1
$0.8 \times 2 = 1.6$ Integer part 1
$0.6 \times 2 = 1.2$ Integer part 1
$0.2 \times 2 = 0.4$ Integer part 0
$0.4 \times 2 = 0.8$ Integer part 0

$$
\begin{aligned}
0.8 \times 2 &= 1.6 \text{ Integer part } 1 \\
0.6 \times 2 &= 1.2 \text{ Integer part } 1 \\
0.2 \times 2 &= 0.4 \text{ Integer part } 0 \\
0.4 \times 2 &= 0.8 \text{ Integer part } 0 \\
&\ldots \qquad\qquad\qquad \ldots
\end{aligned}
$$

$$0.45_{10} = 0.01\ 1100\ 1100\ \ldots\ 1100_2$$

When the binary representation corresponds to an infinite sequence, one criterion to determine the number of bits needed may be the precision that must be equivalent in both numeration systems. In the above example, if the absolute error (in decimal) is $\pm 5 \times 10^{-3}$, the expansion in powers of 2^{-n} will then stop at the nth term for which the following condition is verified to be true:

$$2^{-n} \leq 5 \times 10^{-3} \qquad [1.9]$$

Similarly, we have:

$$
\begin{aligned}
2^n &\geq 200 \\
n &\geq \frac{\log(200)}{\log(2)} = 7.64 \simeq 8
\end{aligned}
$$

We can thus stop at the eighth row. Thus:

$$0.45_{10} = 0.01110011_2$$

1.10. Arithmetic operations on binary numbers

Arithmetic operations on binary numbers may be executed in the same way as for decimal numbers.

The addition is the most executed arithmetic operation in digital systems. The subtraction operation is essentially a variant of the addition operation, while multiplication and division operations may be carried out by combining logical functions (AND, OR, shift, etc.) and addition.

1.10.1. *Addition*

In binary representation, we begin by adding bits of lower weight, and the carry that may be obtained when the sum of bits of the same weight exceeds the highest

value that can be represented with one bit, that is 1, is transferred, each time, to the next MSB.

In binary representation, addition is carried out according to the following rules:

$$\begin{aligned} 0+0 &= 0 \\ 0+1 = 1+0 &= 1 \\ 1+1 &= 0 \quad \text{Carry 1} \\ 1+1+1 &= 1 \quad \text{Carry 1} \end{aligned}$$

EXAMPLE 1.14.– Add the numbers 1010 and 1011.

Carrying out the addition operation in binary and decimal, we have:

$$\begin{array}{r} 1011 \\ +\ 0011 \\ \hline 1110 \end{array} \qquad \begin{array}{r} 11 \\ +\ 3 \\ \hline 14 \end{array}$$

The sum is obtained by adding the numbers, each of which is called the *addend*.

In practice, more than two numbers can be added in a digital system by initially determining the sum of the first two numbers, then adding this sum to the third number and so on.

1.10.2. *Subtraction*

In binary representation, the execution of a subtraction operation takes place from the LSBs to the MSBs with the assumption that the number to be subtracted (or the subtrahend) is the smaller of the two operands. The difference is the result obtained upon subtracting the subtrahend from the minuend.

Before subtracting a number (bit at the logic level 1) from another number of lower value (bit at the logic level 0), we add the value of the radix (that is 2) to the latter and a borrow of 1 is then carried over to the next highest bit to be subtracted. The rules governing binary subtraction are:

$$\begin{aligned} 0-0 &= 0 \\ 0-1 &= 1 \quad \text{Borrow 1} \\ 1-0 &= 1 \\ 1-1 &= 0 \end{aligned}$$

EXAMPLE 1.15.– Subtract the number 101 from the number 1010.

The subtraction may be carried out in binary representation and in decimal representation as follows:

1010		10	Minuend
− 0101		− 5	Subtrahend
0101		5	Difference

The difference is obtained by deducting the *subtrahend* from the *minuend*.

In practice, subtraction may be carried out like addition by using 2C representation, which allows for the coding of positive and negative numbers.

1.10.3. *Multiplication*

Multiplication is carried out by forming a partial product for each bit of the multiplier and then adding all the partial products to generate the result. It must be noted that each partial product is shifted one position to the left with respect to the preceding one and the product of two n-bit numbers may possess up to $2n$ bits.

The multiplication table in binary representation can be summarized as follows:

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

EXAMPLE 1.16.– Multiply the number 1101 by 1001.

Executing multiplication in binary representation translates to:

	1101	Multiplicand
×	1001	Multiplier
	1101	First partial product
	0000	Second partial product
	0000	Third partial product
+	1101	Fourth partial product
	1110101	Product

This operation is the equivalent of $13 \times 9 = 117$ in decimal.

By convention, the first factor in a multiplication operation is called the multiplicand and the second is called the multiplier. This distinction is of absolutely no consequence as the multiplication operation is commutative. The product is defined as the result of a multiplication.

Multiplication may be carried out like a succession of addition and shift operations.

1.10.4. *Division*

Division of a binary number (the dividend) by another (the divisor) is carried out by repeatedly deducting the divisor from the dividend until you obtain a difference that is either equal to zero or inferior to the divisor and that represents the remainder. The quotient corresponds to the number of times the divisor is contained in the dividend.

When the dividend is a $2n$-bit number and the divisor is an n-bit number, the quotient may be represented as an n-bit number. Division is executed by comparing the n bits of the divisor with the n LSBs of the dividend. If the divisor is greater than the dividend, no subtraction is performed, the corresponding quotient bit is set to 0, and the divisor is then compared to the $n + 1$ LSBs of the dividend. If, on the other hand, the divisor is less than or equal to the considered dividend bits, a subtraction is carried out and the corresponding quotient bit is set to 1. The comparison process of the divisor continues with the number obtained by lowering the next MSB of the dividend to the right of the previously obtained difference.

EXAMPLE 1.17.– Divide the number 10000100 by 1101.

In binary representation, the division is carried out as follows:

Dividend	10000100	1101	Divisor
	− 1101	1010	Quotient
	0111**1**		
	0111**0**		
	− 1101		
Remainder	01**0**		

In decimal, we similarly have $132 \div 13 = 10$ and the remainder is 2.

An integer number is divisible by another when the quotient is an integer number and the remainder is equal to zero.

1.11. Representation of real numbers

Real numbers are useful in digital systems as they allow for a variety of calculations. They may be represented with a fixed point or floating point. Fixed-point representation allows for coding a fixed range of numbers and rapid calculation, while coding numbers of very different orders of magnitude is easier with floating-point representation.

1.11.1. *Fixed-point representation*

In fixed-point representation, a number may be expressed in the form:

$$b_{q-1}b_{q-2}\cdots b_0, b_{-1}b_{-2}\cdots b_{-p} \qquad [1.10]$$

The sign-bit b_{q-1} is either equal to 0, for a positive number, or 1, for a negative number. The first q numbers represent the integer part while the last p numbers constitute the fractional part.

According to the SM notation, the value of a decimal number represented in the radix B is given by:

$$N_{10} = (-1)^{b_{q-1}} \sum_{i=-p}^{q-2} b_i B^i \qquad [1.11]$$

By setting $p + q = n$, we have:

$$N_{10} = (-1)^{b_{q-1}} \sum_{i=0}^{p+q-2} b_{i-p} B^{i-p} \qquad [1.12]$$

$$= \left((-1)^{b_{n-p-1}} \sum_{i=0}^{n-2} b_{i-p} B^i \right) B^{-p} \qquad [1.13]$$

where n is the total number of bits. Fixed-point representation may thus be considered as representing an integer whose bits are shifted according to a factor, the scale of which depends on the radix. The maximum (minimum) value in a fixed-point representation is obtained by multiplying by a scaling factor the greatest (smallest) integer that can be represented with the same number of bits. Hence, the values that can be represented are of the form:

$$-(B^{n-1} - 1)B^{-p} \leq N_{10} \leq (B^{n-1} - 1)B^{-p} \qquad [1.14]$$

EXAMPLE 1.18.– In fixed-point representation, we may obtain the following conversions:

$$\begin{aligned}
124.37_{10} &= (1\times 10^2)+(2\times 10^1)+(4\times 10^0)+(3\times 10^{-1})+(7\times 10^{-2})\\
11.625_{10} &= (1\times 2^3)+(0\times 2^2)+(1\times 2^1)+(1\times 2^0)+\\
&\qquad (1\times 2^{-1})+(0\times 2^{-2})+(1\times 2^{-3}) = 1011.101_2\\
20.75_{10} &= (2\times 8^1)+(4\times 8^0)+(6\times 8^{-1}) = 24.6_8\\
30.5_{10} &= (1\times 16^1)+(14\times 16^0)+(8\times 16^{-1}) = 1E.8_{16}
\end{aligned}$$

In 2C representation, the decimal value of a number can be expressed as:

$$N_{10} = \left(-b_{n-p-1}\cdot 2^{n-1}+\sum_{i=0}^{n-2} b_{i-p}2^i\right)2^{-p} \qquad [1.15]$$

Using n bits, the range of numbers that may be represented is given by:

$$-2^{n-1}2^{-p} \le N \le (2^{n-1}-1)2^{-p} \qquad [1.16]$$

where the number of bits for the fractional part is equal to p.

EXAMPLE 1.19.– Give the 8-bit representation of the decimal numbers 6.25 and -8.4375.

We have:

$$\begin{aligned}
6.25_{10} &= (0\times 2^3)+(1\times 2^2)+(1\times 2^1)+(0\times 2^0)+(0\times 2^{-1})+(1\times 2^{-2})\\
&= 0110.0100_2 = 0110.0100_{2C}\\
-8.4375_{10} &= -(1\times 2^3)+(0\times 2^2)+(0\times 2^1)+(0\times 2^0)\\
&\qquad +(0\times 2^{-1})+(1\times 2^{-2})+(1\times 2^{-3})+(1\times 2^{-4})\\
&= 1000.0111_{2C}
\end{aligned}$$

The result obtained upon multiplying two n bit numbers must be stored in $2n$ bits. The size of the data may continually increase following the execution of other multiplication operations. As the product of the numbers in the range from -1 to 1 always stays in the same interval, the solution adopted in digital systems consists of using a representation ($q = 1$ and $n = p + 1$) in which the numbers are normalized and can only vary between -1 and 1.

1.11.2. *Floating-point representation*

Floating-point representation may be considered as a scientific notation for digital systems. A certain number of floating-point representations have been proposed in order to satisfy the requirements of a variety of applications.

A decimal number N can be quantified and expressed in floating-point form as follows:

$$N_{10} = (-1)^S M \cdot B^E \qquad [1.17]$$

where S is the sign-bit, M is the mantissa, B is the base or radix and E is the exponent. The mantissa is generally normalized and corresponds to a number beginning with a non-zero digit, as is the case with the following number representations:

-1234.57_{10} is written as -1.23457×10^3;

0.0000071539_{10} is written as $+7.1539 \times 10^{-6}$;

100010100_2 is written as 1.00010100×2^8.

As a result of the normalization of the mantissa, M, the number 0 cannot be represented directly from the expression [1.17]. To arrive at this, we must use a particular symbol. Indefinite numbers, such as the result of a division by 0 or the square root of a negative integer, are also represented using special characters.

1.11.2.1. *IEEE-754 standard*

Norms or standards have been proposed in order to make the different representations of floating-point numbers uniform.

In the IEEE[1]-754 norm, the mantissa M and the exponent E must satisfy the following inequalities:

$$1 \leq M < 2 \qquad [1.18]$$

and

$$2 - 2^{k-1} \leq E \leq 2^{k-1} - 1 \qquad [1.19]$$

1 IEEE: Institute of Electrical and Electronics Engineers.

The binary equivalent of the mantissa M is thus normalized, and the exponent E is written in biased form before being coded as a k-bit word. The values that can be represented for a number N are such that:

$$N_{\min} = 2^{2-2^{k-1}} \leq |N| \leq N_{\max} = (2 - 2^{-l})2^{2^{k-1}-1} \quad [1.20]$$

The parameter l is defined as being the number of bits of the mantissa. Figure 1.7 shows the range of numbers that can be represented in a floating-point format.

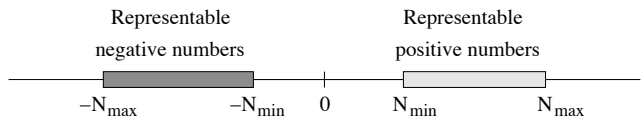


Figure 1.7. *Range of numbers that can be represented in floating-point format*

NOTE 1.4.– As the first digit of the mantissa is always 1, it can be taken as implied. This gives us an additional bit position that can be exploited to increase the range of representable numbers.

The relative difference between two adjacent numbers is of the order 2^{l-l}. It is, therefore, necessary to round off some numbers before representing them.

Precision	Normalized representation	Denormalized representation
Single	$\pm 2^{-126}$ to $(2 - 2^{-23}) \times 2^{127}$	$\pm 2^{-149}$ to $(1 - 2^{-23}) \times 2^{-126}$
Double	$\pm 2^{-1022}$ to $(2 - 2^{-52}) \times 2^{1023}$	$\pm 2^{-1074}$ to $(1 - 2^{-52}) \times 2^{-1022}$

Table 1.3. *Range of numbers that can be represented with the IEEE-754 standard*

The majority of numbers in IEEE-754 floating-point representation are normalized and have a mantissa of the form:

$$M = 1.\underbrace{f_1 f_2 \cdots f_l}_{f}$$

where the fractional part (or fraction) f is represented with l bits, and $1 \leq M < 2$.

As shown in Table 1.4, the IEEE-754 standard defines two formats for number representation: single precision (or 32 bits, composed of 1 sign bit, 8 exponent bits

and 23 mantissa bits) and double precision (or 64 bits, composed of 1 sign-bit, 11 exponent bits and 52 mantissa bits).

	Sign bit	Biased exponent	Mantissa fraction
32 bit single precision	1 bit	8 bits	23 bits
64 bits double precision	1 bit	11 bits	52 bits

Table 1.4. *Number format based on the IEEE-754 standard*

In addition to the single and double precisions, the IEEE-754 standard supports quadruple-precision representation (or 128 bits, composed of 1 sign bit, 15 exponent bits and 112 mantissa bits), which is chiefly used in some software.

When an arithmetic operation involving two numbers gives a result that has an exponent that is too small to be accurately represented, an underflow is produced. The IEEE-754 standard, through the use of denormalized representation, offers a means of gradually taking into account underflows.

A denormalized number is characterized by a biased exponent equal to 0 and a mantissa of the form:

$$M = 0.\underbrace{f_1 f_2 \cdots f_l}_{f}$$

The mantissa bits are shifted one position to the right to insert the first bit (implied in normalized representation), which now has the value 0. To compensate for the shift effect, the exponent is increased by 1.

Table 1.3 gives the range of numbers that can be represented using the IEEE-754 standard.

The exponent E is a signed k-bit integer such that $E_{\min} \leq E \leq E_{\max}$. Its representation corresponds to the representation of the biased value $E + b$, where b is the bias of the form $2^{k-1} - 1$. Furthermore, $E_{\min} = -b + 1$ and $E_{\max} = b$. The exponents $E_{\min} - 1$ and $E_{\max} + 1$ (0 and $2^k - 1$, respectively, in the biased representation) are reserved for zero, denormalized numbers and special values.

To facilitate the coding of positive and negative values of the exponent, a bias, b, is added to the real value of the exponent, E, as follows:

$$E_b = \begin{cases} E + b, & \text{if the number is normalized} \\ E + b - 1, & \text{if the number is denormalized} \end{cases} \quad [1.21]$$

Thus, in the IEEE-754 standard, the exponent corresponds to the binary representation of E_b.

EXAMPLE 1.20.– Represent the decimal numbers 79.625 and -1000.2 in IEEE-754 single precision.

In the IEEE-754 standard, a number is represented by a sign bit, a mantissa and an exponent. The normalized form of the binary equivalent of the number to be converted allows for the identification of the mantissa and the exponent.

The decimal number 79.625 can also be written as follows:

$$79.625_{10} = 1001111.101_2 = 1.001111101_2 \times 2^6$$

– sign bit: $S = 0$;

– biased exponent (8 bits): $E_b = 6_{10} + 127_{10} = 133_{10} = 10000101_2$;

– fractional part of the mantissa (23 bits):

$$f = 00111110100000000000000_2$$

from which:

$$79.625_{10} = 0\ 10000101\ 00111110100000000000000_{IEE754}$$

The decimal number -1000.2 is represented in binary in the form:

$$1000.2_{10} = 1111101000.0011001100110011_2$$

The fractional part corresponds to a continually repeating binary sequence. The closest number to 1000.2 that may be represented is:

$$\begin{aligned} 1000.200012207031250_{10} &= 1111101000.00110011001101_2 \\ &= 1.11110100000110011001101_2 \times 2^9 \end{aligned}$$

– sign bit: $S = 1$;

– biased exponent (8 bits): $E_b = 9_{10} + 127_{10} = 136_{10} = 10001000_2$;

– fractional part of the mantissa (23 bits):

$$f = 11110100000110011001101_2$$

and finally:

$$-1000.2_{10} = 1\ 10001000\ 11110100000110011001101_{IEE754}$$

In the above-cited single-precision IEEE-754 representations, the first bit indicates the sign, the next eight bits allow for the coding of the exponent and the last 23 bits correspond to the fractional part of the mantissa.

The different values taken by the numbers in IEEE-754 representations are recorded in Table 1.5. The IEEE-754 standard uses special symbols (NaN, infinity) to indicate numbers that have an exponent composed entirely of bits set to 0 or 1. The NaN or *not a number* value is used to represent a value that does not correspond to a real number.

	Exponent	Fraction	Value
Normalized	$E_{min} \leq E \leq E_{max}$	$f \geq 0$	$\pm(1.f) \times 2^E$
Denormalized	$E = E_{min} - 1$	$f > 0$	$\pm(0.f) \times 2^{E_{min}}$
Zero	$E = E_{min} - 1$	$f = 0$	± 0
Infinite	$E = E_{max} + 1$	$f = 0$	$\pm\infty$
Not a Number	$E = E_{max} + 1$	$f > 0$	NaN

Table 1.5. *Values of numbers in IEEE-754 representations*

EXAMPLE 1.21.– Find the decimal number corresponding to the following single-precision IEEE-754 representation:

$$1\ 10000111\ 11000000000000000100001_{IEE754}$$

We have:

– sign bit: $S = 1$;

– biased exponent (8 bits): $E_b = 10000111_2 = 135_{10}$;

– fractional part of the mantissa (23 bits):

$$f = 11000000000000000100001_2$$

Applying the formula to the expression of real numbers by starting from the IEEE-754 representation, that is:

$$N_{10} = (-1)^S(1.f) \times 2^{(E_b-127)}$$

we find:

$$\begin{aligned} N_{10} &= (-1)^1(1.11000000000000000100001_2) \times 2^{(135-127)} \\ &= (-1)(111000000.000000000100001_2) \\ &= (-1)(2^8 + 2^7 + 2^6 + 2^{-10} + 2^{-15}) \\ &= -448.00100708_{10} \end{aligned}$$

from which:

$$1\ 10000111\ 11000000000000000000001_{IEE754} = -448.001_{10}$$

1.11.2.2. *Arithmetic operations on floating-point numbers*

Let $x = M_x \cdot B^{E_x}$ and $y = M_y \cdot B^{E_y}$ be two positive numbers (sign-bit $S = 0$).

Supposing that $E_x \geq E_y$, $y = M_Y \cdot B^{E_x}$ and $M_Y = M_y B^{(E_x - E_y)}$, we have:

$$x + y = (M_x + M_Y) \cdot B^{E_x} \qquad [1.22]$$

and

$$x - y = (M_x - M_Y) \cdot B^{E_x} \qquad [1.23]$$

In a floating-point representation, the numbers to be added or subtracted must, thus, have the same exponent, such as:

$$\begin{aligned} 145.500_{10} &= 10010001.100_2 = 0.10010001100 \times 2^8 \\ 27.625_{10} &= 00011000.101_2 = 0.00011011101 \times 2^8 \end{aligned}$$

In the case of multiplication and division, the results are obtained as follows:

$$x \times y = (M_x \times M_y) \cdot B^{(E_x + E_y)} \qquad [1.24]$$

and

$$x/y = (M_x/M_y) \cdot B^{(E_x - E_y)} \qquad [1.25]$$

It must be noted that because of the overflow effect or round-off errors, the arithmetic operations in a floating-point representation do not have exactly the same properties (associativity, distributivity) as with real numbers.

1.12. Data representation

As the arithmetic unit of a digital system recognizes only the binary states 0 and 1, a code is necessary to manipulate and transfer alphanumeric data (numbers, letters, special characters) between a digital system and its peripheral devices.

1.12.1. *Gray code*

Gray code (or reflected binary code) is a non-weighted code, as it does not ascribe a specific weight to each bit position. It is not used for arithmetic calculations.

An interesting feature presented by Gray code representation is related to the fact that only a single bit changes value during the transition from one code to the next. Table 1.6 gives the binary and Gray code representation of decimal numbers from 0 to 15.

The conversion of a binary number to Gray code is carried out by making use of the following observations:

– the most significant Gray code bit, situated to the extreme left, is the same as the corresponding MSB for the binary number;

– starting from the left, add, without taking into account the carry-out bit, each pair of adjacent bits to obtain the next bit in Gray code.

EXAMPLE 1.22.– Convert the binary number 11001_2 to Gray code.

Binary number	1	+ 1	+ 0	+ 0	+ 1
Gray code	1	0	1	0	1

For the binary number 11001_2, the corresponding Gray code is 10101.

To convert Gray code to a binary number:

– the MSB of the binary number, located at the extreme left, is identical to the corresponding Gray code bit;

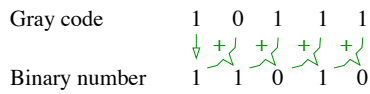
– starting from the left, add each new bit of the binary code to the next bit of the Gray code, without taking into account any carry-out bit, to obtain the next bit of the binary code.

Decimal number	Binary number	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

Decimal number	Binary number	Gray code
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Table 1.6. *Binary and Gray code representation of decimal numbers from* 0 *to* 15

EXAMPLE 1.23.– Convert the Gray code 10111 to a binary number.



The binary number corresponding to Gray code 10111 is 11010_2.

Gray code is used in Karnaugh maps and in the design of logic circuits. They also find application in rotary encoders, where the predisposition to errors increases with the number of bits that change logical states between two consecutive positions.

1.12.2. *p-out-of-n code*

A *p-out-of-n code* is an n-bit representation that allows only combinations made up of p bits at 1 and $(n-p)$ bits at 0. The number of valid combinations for a p-out-of-n code is $n!/[(n-p)!p!]$.

The p-out-of-n code allows for the detection of errors based on the verification of the number of 1s and 0s at the time of reading of each code combination.

Some barcodes use p-out-of-n encoding, such as 2-out-of-5 encoding. Table 1.7 offers some examples of 2-out-of-5 code. These two codes are weighted only for numbers different from zero and the list of weights appears in each of the denominations.

The 2-out-of-5 code allows for the detection of all errors relating to a single bit, but does not allow for the correction of these errors. As the smallest Hamming

distance (or the minimum number of bits that change logic states between two consecutive combinations) is 2, it does not allow for the detection of errors caused by the modification of 2 bits.

	2-out-of-5 code					2-out-of-5 code				
	0	1	2	3	6	7	4	2	1	0
0	0	1	1	0	0	1	1	0	0	0
1	1	1	0	0	0	0	0	0	1	1
2	1	0	1	0	0	0	0	1	0	1
3	1	0	0	1	0	0	0	1	1	0
4	0	1	0	1	0	0	1	0	0	1
5	0	0	1	1	0	0	1	0	1	0
6	1	0	0	0	1	0	1	1	0	0
7	0	1	0	0	1	1	0	0	0	1
8	0	0	1	0	1	1	0	0	1	0
9	0	0	0	1	1	1	0	1	0	0

Table 1.7. *Examples of 2-out-of-5 code*

Barcodes used to sort out letters are represented as shown in Figure 1.8(a), by a series of parallel lines of variable size. The 0 bit corresponds to a small line and the 1 bit to a large line. Figure 1.8(b) shows another barcode that is used to identify parts and that is composed of parallel lines of variable thickness. The 0 bit is represented by a fine line and the 1 bit by a thick line.

(a) (b)

Figure 1.8. *Barcodes corresponding to the binary representation* 01100

A more compact form of the barcode is obtained by using interleaved 2-out-of-5 encoding. The first code is represented by the black lines (three fine lines and two thick lines) of variable thickness, and the second code by the spacing between the black lines (three narrow spaces and two wide spaces). The code shown in Figure 1.9(a) is a representation of the combination 01100 (black lines) followed by 11000 (spaces between the back lines). In general, the odd combinations are represented by black lines and the even combinations are represented by spaces between the black lines. Figure 1.9(b) shows the barcode corresponding to the sequence 01100, 11000, 10001 and 00110.

An appropriate optical reader is necessary to read each kind of barcode.

Figure 1.9. *Barcodes based on an interleaved* 2*-out-of-*5 *encoding*

1.12.3. *ASCII code*

ASCII code (or *American standard code for information interchange*) has seven bits allowing for the representation of $2^7 = 128$ symbols.

Table 1.8 gives the correspondence between certain characters and the decimal and hexadecimal numbers of the ASCII code. The letter *N*, for example, is represented in ASCII code by the number 78 in decimal and by $4E$ in hexadecimal. The ASCII code contains 34 characters used to define the format of information and the space between data and to control the transmission and reception of symbols.

1.12.4. *Other codes*

Given the ever-increasing number of characters, other systems of data representation were developed based on the ASCII code:

– EBCDIC (or *extended binary coded decimal interchange code*) is an eight bit code;

– ANSI (or *American national standard institute*) allows for the representation of alphabetical letters from many languages;

– using eight bit words (for UTF-8), 16 bit words (for UTF-16) and 32 bit words (for UTF-32), the universal code, named Unicode (or *Universal code*) represents each character in a unique way by a number. It covers symbols used in most languages.

1.13. Codes to protect against errors

There are different types of codes used to detect and correct errors that come up in digital information during transmission or during storage.

1.13.1. *Parity bit*

To facilitate the detection of errors, a supplementary bit or parity bit is often added at the end of a binary word with a fixed number of bits. It allows for the allocation of an odd or even parity depending on whether the total number of 1 bits in the code is odd or even.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20	SP	64	40	@	96	60	‘
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	’	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	NP	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

Table 1.8. *ASCII codes table*

EXAMPLE 1.24.– For the word 0101101, the parity bit is 0 (even parity: 4 bits at 1).

For the word 1010001, the parity bit is 1 (odd parity: 3 bits at 1).

Using a single parity bit allows for the detection of all errors that affect only one bit. However, it does not allow for the correction of these errors.

1.13.2. *Error correcting codes*

The reliability of data transmission is generally ensured by using more elaborate codes.

1.13.2.1. *Block codes*

In the block code approach, a certain number of control bits are appended to the message that is structured in blocks of fixed size. In this way, horizontal and vertical parity of data can be verified.

Hamming distance corresponds to the number of bits that vary between two successive words.

EXAMPLE 1.25.– There is a Hamming distance of 3 between the words 111011 and 101010. At least three errors are necessary to make these two words identical.

A possible way of increasing the Hamming distance of a code consists of using several control bits. In this case, a message comprises m bits of data and k control bits.

EXAMPLE 1.26.– Represent OUI in ASCII code with odd (horizontal and vertical) parity bits and a crossed parity bit allowing for the indication of the integrity of the (horizontal and vertical) parity bits.

The ASCII codes for the characters of the word OUI are as below:

$$\begin{aligned} 79_{10} &= 4F_{16} = 1001111_2 \text{ for O} \\ 85_{10} &= 55_{16} = 1010101_2 \text{ for U} \\ 73_{10} &= 49_{16} = 1001001_2 \text{ for I} \end{aligned}$$

The choice of a two-dimension representation (or a block of bits), as shown in Figure 1.10, allows for the definition of parity bits following the horizontal and vertical direction.

Changing one single bit of the data may bring about a modification of the vertical parity bit, the horizontal parity bit and the crossed parity bit, that is four bits in total. The Hamming distance is, thus, equal to 4.

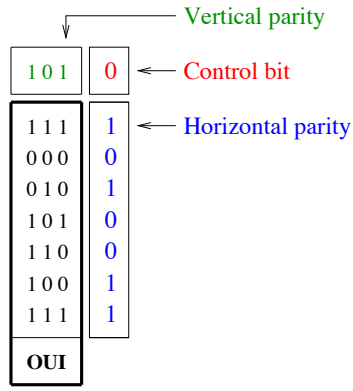


Figure 1.10. *Example of block codes*

Such a block code allows for the detection and correction of all errors affecting one single bit. It allows for the detection of all errors affecting 2 and 3 bits, but it presents the inconvenience of requiring the verification of a large number of bits.

1.13.2.2. *Cyclic codes*

Cyclic codes are based on the transcription of binary numbers in polynomial form and the division of polynomials.

EXAMPLE 1.27.– The binary code $b_{n-1}b_{n-2}\ldots b_1b_0$ corresponds to the polynomial:

$$b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1x^1 + b_0x^0$$

Let $I(x)$ be the polynomial associated with a message. Supposing that $G(x)$ is an r generator polynomial, the message may be coded by carrying out the following operations:

– multiply $I(x)$ by x^r (or add r zeros at the end of $I(x)$);

– decompose $I(x)x^r$ into the form:

$$\frac{I(x)x^r}{G(x)} = Q(x) + R(x) \qquad [1.26]$$

– determine the cyclic polynomial $T(x)$:

$$T(x) = I(x)x^r - R(x) \qquad [1.27]$$

The polynomial $T(x)$ is a multiple of $G(x)$. It corresponds to a representation of data to which redundant bits have been appended.

Errors are detected by verifying the divisibility of $T(x)$ by $G(x)$.

NOTE 1.5.– Expressions used for the generator polynomials vary by application areas:

– CRC[2]-3-GSM: $G(x) = x^3 + x + 1$;

– CRC-4-ITU: $G(x) = x^4 + x + 1$;

– CRC-8-CCITT: $G(x) = x^8 + x^2 + x + 1$;

– CRC-16-CCITT: $G(x) = x^{16} + x^{12} + x^5 + 1$;

– CRC-32-IEEE: $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$;

– CRC-64-ISO: $G(x) = x^{64} + x^4 + x^3 + x + 1$.

EXAMPLE 1.28.– Let us consider the initial information 101101, with which the polynomial $I(x) = x^5+x^3+x^2+1$ can be associated. Using the polynomial generator of the form $G(x) = x^3 + x + 1$ ($r = 3$), the form of the word to be transmitted, or the polynomial $T(x)$, is determined by proceeding as per the steps:

– multiplication of $I(x)$ by x^r gives the product $I(x)x^r = 101101000$;

– the division of $I(x)x^r$ by $G(x)$ yields the quotient $Q(x) = 100001$ and the remainder $R(x) = 011$;

– the polynomial $T(x)$ is finally obtained by appending the r bits of $R(x)$ to the end of $I(x)$, that is: $T(x) = 101101011$.

In the form $T(x) + E(x)$, the information is assumed to be affected by the error E. With a code based on a polynomial generator $G(x)$, we can detect:

– all the single errors ($E = 10\ldots0$);

– all the double errors ($E = 10\ldots010\ldots0$) if $G(x)$ has a factor with at least three terms;

– all the errors relating to an odd number of bits (E has an odd number of bits at 1) if $x + 1$ divides $G(x)$;

– all the series of errors ($E = 0\ldots01\ldots10\ldots0$) of length smaller than the degree of $R(x)$;

– most of the long series of errors.

2 CRC: cyclic redundancy check.

1.14. Exercises

EXERCISE 1.1.– Conversions

1) Convert the following numbers to binary:

a) 37_{10} b) 15_{10} c) 187_{10} d) $2\,014_{10}$ e) $2\,016_{10}$ f) 2.75_{10}
g) 25.25_{10} h) 243.3125_{10} i) 0.0625_{10} j) 62_8 k) 277_8 l) 12.6_8
m) 476.35_8 n) 92_{16} o) $37FD_{16}$ p) $7FF_{16}$ q) $1A6_{16}$ r) $2C0_{16}$
s) $1F.C_{16}$ t) $9.F_{16}$ u) $A7.EC_{16}$

2) Convert the following numbers to decimal:

a) 10110_2 b) 10001_2 c) 10001101_2 d) 100100001001_2 e) 1111010111_2
f) 1011.101_2 g) 10011011001.10110_2 h) 30_8 i) 115_8 j) 55.4_8
k) 270.54_8 l) 356_{16} m) $2AF_{16}$ n) $2C1_{16}$ o) $10FF_{16}$
p) $1FCFA_{16}$ q) $DADA.C_{16}$ r) $F.4_{16}$ s) $EBA.C_{16}$

3) Convert the following numbers to hexadecimal:

a) 320_{10} b) $6\,861_{10}$ c) $65\,535_{10}$ d) 100_8 e) 62.4_8 f) 500.25_8
g) 10001101_2 h) 1001000110100011110_2 i) 10000.1_2
j) 1000000.0000111_2 k) 1000111001.01_2

4) Convert the following BCD numbers to decimal:

a) $0001\ 1000\ 0100_{BCD}$ b) $0100\ 1001\ 0010_{BCD}$
c) $1001\ 0111\ 0101\ 0010_{BCD}$ d) $0111\ 0111\ 0111\ 0101_{BCD}$

5) How many bits are required for the binary representation of the decimal numbers from 0 to 511?

6) What is the largest number that can be represented in 16-bit binary numeration system?

7) a) Determine the binary representation of the decimal number 10.05 with an absolute error equal to 0.005.

b) Represent the decimal number 0.452 in binary numeration system with a relative error of 0.1%.

EXERCISE 1.2.– Representation of numbers and data

1) Let X be an unsigned n-bit integer. Verify that $2^n - X$ represents the two's complement of $-X$.

2) Determine the 8-bit two's complement of the following numbers: -1_{10}, -17_{10}, -128_{10}.

Convert the following numbers to decimal:

01111111_{2C}, 11001110_{2C}, 10001000_{2C}.

3) Determine the two's complement representations of the numbers $-63A_{16}$ and $-8AC_{16}$.

4) Convert the following numbers to ASCII codes:

a) 1_{10}, b) 107_{10}, c) 1000010_2.

5) Use ASCII code to translate the following expressions:

$X = \cos(.7)$
$Y = 256 * X$
PRINT "X=", X; "Y=", Y

6) a) Represent the following numbers in single-precision IEEE-754 standard: 2.75_{10}, $-417\ 680_{10}$.

b) Find the decimal number corresponding to each of the following single-precision IEEE-754 representations:

$0\ 10001010\ 01110111000110000000000_{IEE754}$
$00000000\ 00000000000000000000001_{IEEE754}$

EXERCISE 1.3.– Gray code/binary number conversion

1) Convert the following binary numbers to Gray code:

a) 11011_2 b) 101101_2 c) 11000110_2

2) Convert each of the following Gray code to binary number:

a) 1010_{Gray} b) 00010_{Gray} c) $11000010001_{\text{Gray}}$

EXERCISE 1.4.– Correction code

1) Data are to be transmitted, coded by the CRC method, whose polynomial generator is:

$$G(x) = x^3 + x + 1$$

Determine the bits of the message to be transmitted if the initial message is 101101.

What can we say of the transmission if the received message is 100101 011?

2) Using the CRC coding method, represent the message to be transmitted in the case where the initial information is 1011001 and the polynomial generator is of the form:

$$G(x) = x^4 + x + 1$$

1.15. Solutions

SOLUTION 1.1.– Conversions

1) Conversions to binary representation

a) $37_{10} = 100101_2$

b) $15_{10} = 1111_2$

c) $187_{10} = 10111011_2$

d) $2\ 014_{10} = 11111011110_2$

e) $2\ 016_{10} = 2^{11} - 2^5 = 11111100000_2$

f) $2.75_{10} = 10.11_2$

g) $25.25_{10} = 11001.01$

h) $243.3125_{10} = 11110011.0101_2$

i) $0.0625_{10} = 0.0001_2$

j) $62_8 = 110010_2$

k) $277_8 = 10111111_2$

l) $12.6_8 = 1010.11_2$

m) $476.35_8 = 100111110.011101_2$

n) $92_{16} = 10010010_2$

o) $37FD_{16} = 11011111111101_2$

p) $7FF_{16} = 11111111111_2$

q) $1A6_{16} = 110100110_2$

r) $2C0_{16} = 1111000000_2$

s) $1F.C_{16} = 11111.11_2$

t) $9.F_{16} = 1001.1111_2$

u) $A7, EC_{16} = 10100111.111011_2$

2) Conversion to decimal representation

a) $10110_2 = 22_{10}$

b) $10001_2 = 17_{10}$

c) $10001101_2 = 141_{10}$

d) $100100001001_2 = 2313_{10}$

e) $1111010111_2 = 983_{10}$

f) $1011.101_2 = 11.625_{10}$

g) $10011011001.10110_2 = 1241.6875_{10}$

h) $30_8 = 36_{10}$

i) $115_8 = 77_{10}$

j) $55.4_8 = 45.5_{10}$

k) $270.54_8 = 184.6875_{10}$

l) $356_{16} = 854_{10}$

m) $2AF_{16} = 687_{10}$

n) $2C1_{16} = 705_{10}$

o) $10FF_{16} = 4351_{10}$

p) $1FCFA_{16} = 130298_{10}$

q) $DADA.C_{16} = 56026.75_{10}$

r) $F.4_{16} = 15.25_{10}$

s) $EBA.C_{16} = 3770.75_{10}$

3) Conversion to hexadecimal representation

a) $320_{10} = 140_{16}$

b) $6\ 861_{10} = 1ACD_{16}$

c) $65\ 535_{10} = 16^4 - 1 = FFFF_{16}$

d) $100_8 = 40_{16}$

e) $62.4_8 = 32.8_{16}$

f) $500.25_8 = 140.54_{16}$

g) $10001101_2 = 8D_{16}$

h) $1001000110100011110_2 = 48D1E_{16}$

i) $10000.1_2 = 10.8_{16}$

j) $1000000.0000111_2 = 40.0E_{16}$

k) $1000111001.01_2 = 239.4_{16}$

4) BCD – Decimal conversion

a) $0001\ 1000\ 0100_{BCD} = 184_{10}$

b) $0100\ 1001\ 0010_{BCD} = 492_{10}$

c) $1001\ 0111\ 0101\ 0010_{BCD} = 9\ 752_{10}$

d) $0111\ 0111\ 0111\ 0101_{BCD} = 7\ 775_{10}$

5) How many bits are required for the binary representations of the decimal numbers from 0 to 511?

With k bits, only the decimal numbers from 0 to $2^k - 1$ can be represented. Thus:

$$2^k - 1 = 511 \quad \text{and} \quad k = \log(512)/\log(2) = 9$$

6) What is the largest number that can be represented in 16 bit binary numeration?

The largest number that can be represented in 16 bits binary numeration system is $2^{16} - 1 = 65.535$.

7) a) Determine the binary representation of the decimal number 10.05 with an absolute error equal to 0.005.

To ensure that the difference between the represented value and the decimal number 10.05 remains less than or equal to 0.005, the required number of bits, n, must be determined based on the following relationship:

$$2^{-n} \leq 0.005$$

that is:

$$n \geq \log(1/0.005)/\log(2) = 7.64 \simeq 8$$

and finally we have:

$$10.05_{10} = 1010.00001100_2$$

b) Represent the decimal number 0.452 in binary with a relative error of 0.1%.

The desired value of the absolute error is $0.001 \times 0.452 = 0.000452$.

The binary representation must possess a number of bits, n, that satisfies the following equation:

$$2^{-n} < 0.000452$$

that is:

$$n > \log(1/0.000452)/\log(2) = 11.11 \simeq 12$$

Thus:

$$0.452_2 = 0.0111001110110110_2$$

SOLUTION 1.2.– Representation of numbers and data

1) Let X be an unsigned n-bit integer. Verify that $2^n - X$ represents the 2C of $-X$.

Let $X = X_{n-1}X_{n-2}\cdots X_1X_0$ be the n-bit binary representation of a positive number. We can write:

$$2^n - X = (2^n - 1) - X + 1$$

where:

$$2^n - 1 = \underbrace{111\cdots11}_{n \text{ bits}}{}_2$$

The following subtraction operation:

$$\underbrace{111\cdots11}_{n \text{ bits}}{}_2 - X_{n-1}X_{n-2}\cdots X_1X_0$$

is equal to the inversion of the logical level of the bits of X. Thus:

$$2^n - X = (2^n - 1) - X + 1 = \overline{X} + 1$$

is the 2C representation of $-X$.

2) Determine the 8-bit 2C of the following numbers: -1_{10}, -17_{10}, -128_{10}.

For each of the numbers, we can obtain:

$$1_{10} = 00000001_2$$
$$11111110_{1C} + 1 = 11111111_{2C} = -1_{10}$$

$$17_{10} = 00010001_2$$
$$11101110_{1C} + 1 = 11101111_{2C} = -17_{10}$$

$$128_{10} = 10000000_2$$
$$01111111_{1C} + 1 = 10000000_{2C} = -128_{10}$$

3) Convert the following numbers to decimal representation: 01111111_{2C}, 11001110_{2C} and
10001000_{2C}.

The conversions are carried out as follows:

$$01111111_{2C} = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 127_{10}$$

$$11001110_{2C} = -2^7 + 2^6 + 2^3 + 2^2 + 2^1 = -50_{10}$$

$$10001000_{2C} = -2^7 + 2^3 = -120_{10}$$

4) Determine the 2C of the numbers $-63A_{16}$ and $-8AC_{16}$.

We can proceed as follows:

$$\begin{array}{rl} 63A_{16} = & 011000111010_2 \\ & 100111000101_{1C} \\ + & \qquad\qquad 1 \\ \hline = & 100111000110_{2C} = -63A_{16} = 9C6_{16} \end{array}$$

$$\begin{array}{rl} 8AC_{16} = & 0000100010101100_2 \\ & 1111011101010011_{1C} \\ + & \qquad\qquad\qquad 1 \\ \hline = & 1111011101010100_{2C} = -8AC_{16} = F754_{16} \end{array}$$

5) Convert the following numbers to ASCII code:

a) 1_{10} b) 107_{10} c) 1000010_2

The corresponding ASCII codes are as follows:

a) 1_{10} : SOH (start of heading) b) $107_{10} : k$

c) $1000010_2 = 66_{10}$: B

6) Use ASCII code to translate the following expressions:

$X = \cos(.7)$
$Y = 256 * X$
PRINT "X=", X; "Y=", Y

	ASCII code	
Symbol	Dec.	Hex.
X	88	58
=	61	3D
c	99	63
o	111	6F
s	115	73
(	40	28
.	46	2E
7	55	37
)	41	29
Y	89	59
=	61	3D
2	50	32
5	53	35
6	54	36
*	42	2A
X	88	58

	ASCII code	
Symbol	Dec.	Hex.
P	80	50
R	82	52
I	73	49
N	78	4E
T	84	54
Espace	32	20
"	34	22
X	88	58
=	61	3D
"	34	22
,	44	2C
X	88	58
"	34	22
Y	89	59
=	61	3D
"	34	22
,	44	2C
Y	89	59

7) a) Represent the following numbers in single-precision IEEE-754 standard: 2.75_{10} and $-417\ 680_{10}$.

The binary conversion of 2.75_{10} yields:

$$2.75_{10} = 10.11_2$$

The normalized form is written as:

$$10.11_2 = 1.011_2 \times 2^1$$

We thus have:

– the mantissa $M = 1.f$ (f represents the 23 bit fractional part):

$f = 01100000000000000000000$

– the exponent of 8 bits:

$E_b = E + b = 1_{10} + 127_{10} = 128_{10} = 10000000_2$

– the sign bit:

$S = 0$ (in the case of a positive number)

From which:

$$2.75_{10} = 0\ 10000000\ 01100000000000000000000$$

For the decimal number $-417\ 680$, we obtain a binary representation of the following form:

$$417\ 680_{10} = 1100101111110010000_2$$

The corresponding normalized form can be expressed as:

$$1100101111110010000_2 = 1.100101111110010000_2 \times 2^{18}$$

We thus have:

– the mantissa $M = 1.f$ (f representing the 23 bit fractional part):

$f = 10010111111001000000000$

– the exponent of 8 bits:

$E_b = E + b = 18_{10} + 127_{10} = 145_{10} = 10010001_2$

– the sign bit:

$S = 1$ (in the case of a negative number)

And finally:

$$-417\ 680_{10} = 1\ 10010001\ 10010111111001000000000_{IEEE754}$$

b) Find the decimal number corresponding to each of the following single-precision IEEE-754 representations:

$$0\ 10001010\ 01110111000110000000000_{IEE754}$$

$$1\ 00000000\ 00000000000000000000001_{IEEE754}$$

For $N_{10} = 0\ 10001010\ 01110111000110000000000_{IEE754}$, we have:

– the sign bit:

$S = 0$ (in the case of a positive number)

– the exponent of 8 bits:

$E_b = 10001010_2 = 138_{10}$ (normalized number)

and:

$$E = E_b - b = 138 - 127 = 11_{10}$$

– the mantissa M of 23 bits:

$$\begin{aligned} M &= 1.f \\ &= 1.01110111000110000000000_2 \\ &= 2^0 + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-6} + 2^{-7} + 2^{-8} + 2^{-12} + 2^{-13} \\ &= 1.4652099609375_{10} \end{aligned}$$

Hence:

$$N_{10} = (-1)^S M \times 2^E = 1.4652099609375 \times 2^{11} = 3000.75_{10}$$

In the case of $N_{10} = 1\ \ 00000000\ \ 00000000000000000000001_{IEEE754}$, we obtain:

the sign bit:

$S = 1$ (for a negative number)

the exponent of 8 bits:

$$E_b = 00000000_2 = 0_{10} \quad \text{(denormalized number)}$$

and:

$$E = E_b - b + 1 = 0 - 127 + 1 = -126_{10}$$

the mantissa M of 23 bits:

$$\begin{aligned} M &= 0.f \\ &= 0.00000000000000000001 \\ &= 2^{-23} \end{aligned}$$

and finally:

$$N_{10} = (-1)^S M \times 2^E = -2^{-23} \times 2^{-126} = -2^{-149} = -1.4 \times 10^{-45}$$

SOLUTION 1.3.– Gray code/binary number conversion

1) Binary–Gray code conversion:

a) $11011_2 = 10110_{\text{Gray}}$ b) $101101_2 = 111011_{\text{Gray}}$

c) $11000110_2 = 10100101_{\text{Gray}}$

2) Gray code–binary number conversion:

a) $1010_{\text{Gray}} = 1100_2$ b) $00010_{\text{Gray}} = 00011_2$
c) $11000010001_{\text{Gray}} = 10000011110_2$

SOLUTION 1.4.– Correction code

1) The principle of the CRC method consists of processing words and codes as binary polynomials.

– *Coding*

In this case, the polynomial generator is of the degree 3 ($r = 3$) and is written as:

$$G(x) = x^3 + x + 1$$

The correspondence between the initial message and a polynomial form is established as follows:

$$101101 \quad \leftrightarrow \quad I(x) = 1 \cdot x^5 + 0 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1$$

The polynomial associated with the initial message is reduced to:

$$I(x) = x^5 + x^3 + x^2 + 1$$

We have:

$$I(x)x^3 = (x^5 + x^3 + x^2 + 1)x^3 = x^8 + x^6 + x^5 + x^3$$

The division of the polynomial $I(x)x^3$ by $G(x)$ is performed as follows:

$$\begin{array}{l|l} x^8 + x^6 + x^5 + x^3 & x^3 + x + 1 \\ \hline x^8 + x^6 + x^5 & x^5 + 1 \quad \leftarrow \text{Quotient} \\ \hline x^3 & \\ x^3 + x + 1 & \\ \hline x + 1 \quad \leftarrow \text{Remainder} & \end{array}$$

The message to be transmitted, $T(x)$, is obtained by concatenating the bits of the initial message, $I(x)$, and the r bits of the division remainder. That is:

101101 011

It must be noted that the division is carried out using the addition modulo 2 for the coefficients of the polynomials $(0+0=0,\ 1+0=1,\ 1+1=0)$.

– *Verification*

For an error-free transmission, the remainder of the division of the received message, $T(x)$, by the polynomial generator, $G(x)$, must be equal to 0.

The polynomial associated with the received message is of the form:

$$101101\ 011 \quad \leftrightarrow \quad x^8+x^6+x^5+x^3+x+1$$

The division is executed as follows:

$$\begin{array}{l|l} x^8+x^6+x^5+x^3+x+1 & x^3+x+1 \\ \hline x^8+x^6+x^5 & x^5+1 \quad \leftarrow \text{Quotient} \\ \hline \quad\quad x^3+x+1 & \\ \quad\quad x^3+x+1 & \\ \hline \quad\quad\quad\quad 0 & \leftarrow \text{Remainder} \end{array}$$

The remainder being 0, this is an error-free transmission.

Considering the other message that is received, we can obtain the following polynomial:

$$100101\ 011 \quad \leftrightarrow \quad x^8+x^5+x^3+x+1$$

The division is carried out as follows:

$$\begin{array}{l|l} x^8+\quad x^5+\quad x^3+\quad x+1 & x^3+x+1 \\ \hline x^8+x^6+x^5 & x^5+x^3+x \quad \leftarrow \text{Quotient} \\ \hline \quad x^6+\quad\quad x^3+\quad x+1 & \\ \quad x^6+\quad x^4+x^3 & \\ \hline \quad\quad\quad x^4+\quad\quad x+1 & \\ \quad\quad\quad x^4+\quad x^2+x & \\ \hline \quad\quad\quad\quad\quad x^2+\quad 1 & \leftarrow \text{Remainder} \end{array}$$

The remainder of the division is not equal to 0, which is the feature of a received message with errors.

2) Coding of the message to be transmitted when the polynomial generator is $G(x)=x^4+x+1$ and the initial message is 1011001.

The initial message can be associated with the polynomial, $I(x)$, given by:

$$I(x) = x^6 + x^4 + x^3 + 1$$

The multiplication of $I(x)$ by x^4 results in:

$$I(x)x^4 = (x^6 + x^4 + x^3 + 1)x^4 = x^{10} + x^8 + x^7 + x^4$$

The division is performed as follows:

$$\begin{array}{lll}
x^{10} + x^8 + x^7 + & x^4 & \left| x^4 + x + 1 \right. \\
\cline{3-3}
x^{10} + \quad x^7 + x^6 & & x^6 + x^4 + x^2 + x \leftarrow \text{Quotient} \\
\cline{1-1}
\quad x^8 + \quad x^6 + & x^4 & \\
\quad x^8 + & x^5 + x^4 & \\
\cline{1-2}
& x^6 + x^5 & \\
& x^6 + \quad x^3 + x^2 & \\
\cline{1-2}
& x^5 + \quad x^3 + x^2 & \\
& x^5 + \quad x^2 + x & \\
\cline{1-2}
& x^3 + \quad x & \leftarrow \text{Remainder}
\end{array}$$

Concatenating the bits of the initial message, $I(x)$, and the four bits of the remainder of the division, the message to be transmitted, $T(x)$, takes the following form:

1011001 1010

2

Logic Gates

2.1. Introduction

Logic functions provide ways to combine different digital signals – or signals that can only take one of two possible levels: low level (0) and high level (1) – based on the laws of Boolean algebra. These laws are applied using logic gates, which can be classified according to the number of available inputs.

Each logic gate has an equivalent electric circuit. However, an electronic logic gate is very different from its electrical equivalent. It is much faster, smaller, and consumes less electric energy.

Figure 2.1 shows the electric circuit that corresponds to the NOT gate. The light-emitting diode comes on when the switch S_1 is opened and goes off when the switch S_1 is closed.

Figure 2.1. *Electric circuit that is the equivalent of the NOT gate*

The electric circuit shown in Figure 2.2 operates as an AND gate. The diode lights up if and only if both switches S_1 and S_2 are closed.

Figure 2.3 shows the electric circuit for the OR gate. The diode comes on if at least one of the switches (S_1 or S_2) is closed.

Digital Electronics 1: *Combinational Logic Circuits*, First Edition. Tertulien Ndjountche.
© ISTE Ltd 2016. Published by ISTE Ltd and John Wiley & Sons, Inc.

Figure 2.2. *Electric circuit corresponding to the AND gate*

Figure 2.3. *Electric circuit corresponding to the OR gate*

The electric circuit corresponding to the XOR gate is illustrated in Figure 2.4. The diode emits visible light when either the switch S_1 or the switch S_2 is closed.

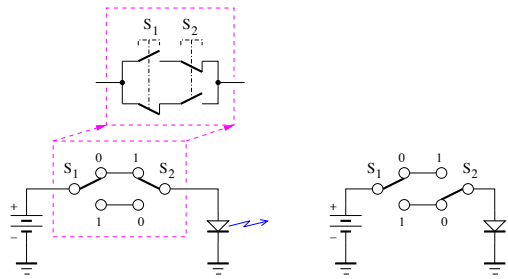


Figure 2.4. *Electric circuit corresponding to the XOR gate, where pressure on either push button S_1 or push button S_2 turns on the diode, but pressure on push button S_1 and push button S_2 turns off the diode*

2.2. Logic gates

Logic gates can be used to combine digital signals based on basic Boolean functions.

2.2.1. *NOT gate*

The NOT function provides the complementary state to a given variable. The function is represented by a bar placed above the input variable and implemented by a NOT gate (or logic inverter).

Figure 2.5. *NOT gate.* $B = \overline{A}$

A	B
0	1
1	0

Table 2.1. *Truth table. Input: A; Output: B*

Figure 2.6 depicts the symbol for a NOT gate. The logic level of the output variable is obtained by taking the complement of the input variable, as shown in the truth table given in Table 2.1. Thus, if an input is at logic level 0, the output will be at logic level 1, and vice versa.

2.2.2. *AND gate*

The AND function, which is also called logic product, is represented by a dot $(\cdot)$.

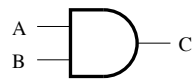


Figure 2.6. *AND gate*
$C = A \cdot B$

The AND gate can have two inputs, as illustrated in Figure 2.6, the output variable takes the high logic level (or the value 1) if and only if the input variables are both at the high logic level (or the value 1). In all other cases, the output is set to the low logic level (or the value 0). Table 2.2 shows the truth table for the AND gate.

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.2. *Truth table.*
Inputs: A, B; Output: C

2.2.3. *OR gate*

The OR function, which is also called logic addition, is represented by a plus $(+)$.

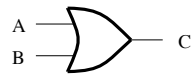


Figure 2.7. *OR gate*
$C = A + B$

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.3. *Truth table.*
Inputs: A, B; output: C

Figure 2.7 depicts the OR gate, which has two gates. As illustrated by the truth table in Table 2.3, the output takes the logic level 1 if at least one of the two inputs is at the logic level 1, it takes logic level 0 if both the inputs are at the logic level 0.

2.2.4. *XOR gate*

The XOR (exclusive OR) function is represented by a plus within a circle $(\oplus)$.

Figure 2.8 depicts the symbolic representation of an XOR gate having two inputs. According to the truth table shown in Table 2.4, the output takes either logic level 1, when only one of the inputs is at logic level 1, or logic level 0, when both inputs are either at logic level 0 or at logic level 1.

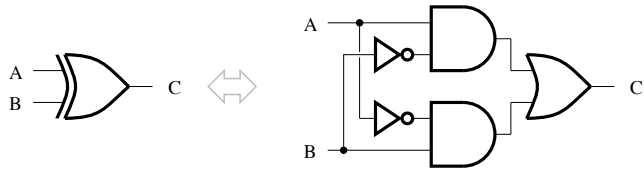


Figure 2.8. *XOR gate (exclusive OR)*
$C = A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.4. *Truth table.*
Inputs: A, B; Output: C

2.2.5. *Complementary logic gates*

The NAND (NOT AND), NOR (NOT OR), and XNOR (NOT exclusive OR or inclusive AND) gates are said to be complementary and correspond, respectively, to the AND, OR and XOR gates when followed by a NOT gate. They are characterized by the following logic equations:

– NAND gate: $C = \overline{A \cdot B}$;

– NOR gate: $C = \overline{A + B}$;

– XNOR gate: $C = \overline{A \oplus B} = A \cdot B + \overline{A} \cdot \overline{B} = A \odot B$.

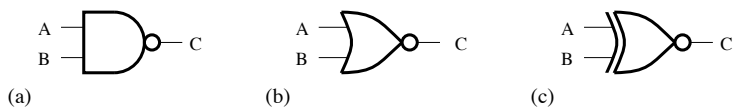


Figure 2.9. *NAND (NOT AND) (a), NOR (NOT OR) (b) and XNOR (NOT exclusive OR) (c) gates*

The NAND and NOR gates are considered to be universal gates. This means that any logic function can be implemented using just NAND or NOR gates. It must be noted that neither the XOR gate nor the XNOR gate is universal.

2.3. Three-state buffer

A three-state buffer works as a signal-controlled switch. An enable signal is used to control whether the input signal is transferred toward the output or isolated from the output, which is then held in a high-impedance state.

The output from the circuit shown in Figure 2.10 can take any of the following three states: high (1), low (0) and high impedance (z).

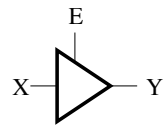


Figure 2.10. *Three-state buffer*

When $E = 0$, the output is held in the high impedance state.

When $E = 1$, the output is at the same state as the input (0 or 1).

Table 2.5 shows the truth table for the three-state buffer.

E	X	Y
0	x	z
1	0	0
1	1	1

Table 2.5. *Truth table*

Using the three-state buffer makes it possible to link the outputs of logic gates set up in parallel through a common line.

2.4. Logic function

A logic function is completely defined when, for all combinations of input variables, the function value is defined. The number of these combinations is 2^n for n variables.

A function is incompletely defined when there is at least one combination of input variables for which the logic level is unknown.

When a logic system is implemented, two cases may occur:

– one of the possible combinations will never exist in the normal functioning of the system. This is called a *forbidden condition* and is denoted by **-** (hyphen);

– one of the combinations exists but can take either the state 0 or 1. This is called the *do not care condition* and is represented by the symbol x or ϕ.

In general, a logic function can thus take four states: 0, 1, x and **-**.

A function with n variables may be represented by a truth table having $n + 1$ columns and a maximum of 2^n lines.

2.5. The correspondence between a truth table and a logic function

Let $X(A, B, C)$ be a logic function with three variables, which is defined by a truth table.

DEFINITION 2.1.– *Based on the truth table in Table 2.6, the function X may be written as the following sum of products:*

$$X(A,B,C) = \overline{A}\cdot\overline{B}\cdot\overline{C} + \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot\overline{C} + A\cdot B\cdot\overline{C} + A\cdot B\cdot C$$
$$= \sum m(0,3,4,6,7) \qquad [2.1]$$

	A	B	C	X	
0	0	0	0	1	$\leftrightarrow \overline{A}\cdot\overline{B}\cdot\overline{C}$
1	0	0	1	0	
2	0	1	0	0	
3	0	1	1	1	$\leftrightarrow \overline{A}\cdot B\cdot C$
4	1	0	0	1	$\leftrightarrow A\cdot\overline{B}\cdot\overline{C}$
5	1	0	1	0	
6	1	1	0	1	$\leftrightarrow A\cdot B\cdot\overline{C}$
7	1	1	1	1	$\leftrightarrow A\cdot B\cdot C$

Table 2.6. *Truth table (sum of products)*

With three variables, $\overline{A}\cdot B\cdot C$ corresponds to a minterm, while $\overline{A}\cdot B$ is not a minterm. A minterm must contain all function variables.

On referring to the truth table in Table 2.7, the product-of-sums form of the function X can be obtained as follows:

$$X(A,B,C) = (A + B + \overline{C})\cdot(A + \overline{B} + C)\cdot(\overline{A} + B + \overline{C})$$
$$= \prod M(1,2,5) \qquad [2.2]$$

	A	B	C	X	
0	0	0	0	1	
1	0	0	1	0	$\leftrightarrow A + B + \overline{C}$
2	0	1	0	0	$\leftrightarrow A + \overline{B} + C$
3	0	1	1	1	
4	1	0	0	1	
5	1	0	1	0	$\leftrightarrow \overline{A} + B + \overline{C}$
6	1	1	0	1	
7	1	1	1	1	

Table 2.7. *Truth table (product of sums)*

With three variables, $A + B + \overline{C}$ represents a *maxterm*, while $B + \overline{C}$ is not a *maxterm*.

The *canonical form* corresponds to the Boolean expression of a logic function using only *minterms* or *maxterms*. It is unique for each logic function.

The *complement* of the function X is given by:

$$\overline{X}(A, B, C) = \prod M(0, 3, 4, 6, 7) = \sum m(1, 2, 5) \qquad [2.3]$$

In the case of four variables, the logic function X is considered to be defined by the truth table in Table 2.8.

The function X can be written as the following sum of products:

$$\begin{aligned} X(A, B, C, D) &= \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot D \\ &= \sum m(1, 5, 7, 15) \end{aligned} \qquad [2.4]$$

Generally, the following relationships exist between the *minterms*, m_i, and the *maxterm*, M_i, of a logic function of n variables:

$$\overline{m_i} = M_{2^n-1-i} \quad \text{or} \quad \overline{M_i} = m_{2^n-1-i} \quad (0 \leq i \leq 2^n - 1) \qquad [2.5]$$

Let n be the number of variables of a logic function:

– the sum of all the 2^n *minterms* of a function is equal to 1;

– the product of all the 2^n *maxterms* of a function is equal to 0.

	A	B	C	D	X	
0	0	0	0	0	0	
1	0	0	0	1	1	$\leftrightarrow \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D$
2	0	0	1	0	0	
3	0	0	1	1	0	
4	0	1	0	0	0	
5	0	1	0	1	1	$\leftrightarrow \overline{A} \cdot B \cdot \overline{C} \cdot D$
6	0	1	1	0	0	
7	0	1	1	1	1	$\leftrightarrow \overline{A} \cdot B \cdot C \cdot D$
8	1	0	0	0	0	
9	1	0	0	1	0	
10	1	0	1	0	0	
11	1	0	1	1	0	
12	1	1	0	0	0	
13	1	1	0	1	0	
14	1	1	1	0	0	
15	1	1	1	1	1	$\leftrightarrow A \cdot B \cdot C \cdot D$

Table 2.8. *Truth table (sum of products)*

The product of two different minterms is equal to 0, while the sum of two different maxterms is equal to 1.

2.6. Boolean algebra

Boolean algebra is applied to operations and functions on logic variables.

Let X and Y be logic (or Boolean) functions, whose values can only be 0 or 1. The following properties are verified:

1) commutativity: $X + Y = Y + X$ and $X \cdot Y = Y \cdot X$;

2) associativity: $X + (Y + Z) = (X + Y) + Z$ and $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$;

3) distributivity: $X \cdot (Y + Z) = X \cdot Y + X \cdot Z$ and $(X + Y)(X + Z) = X + Y \cdot Z$;

4) $\overline{X + Y} = \overline{X} \cdot \overline{Y}$ (DeMorgan's theorem – NOR);

5) $\overline{X \cdot Y} = \overline{X} + \overline{Y}$ (DeMorgan's theorem – NAND).

EXAMPLE 2.1.– Implement an XOR gate from NAND logic gates and an XNOR gate from NOR logic gates.

The logic equation for the XOR gate is given by:

$$C = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B \qquad [2.6]$$

and

$$C = \overline{\overline{C}} = \overline{\overline{A \cdot \overline{B}} \cdot \overline{\overline{A} \cdot B}} \qquad [2.7]$$

Because $A \cdot \overline{B} = A \cdot \overline{A \cdot B}$ and $\overline{A} \cdot B = \overline{A \cdot B} \cdot B$, equation [2.7] takes the form:

$$C = \overline{\overline{A \cdot \overline{A \cdot B}} \cdot \overline{\overline{A \cdot B} \cdot B}} \qquad [2.8]$$

Equation [2.8] can then be implemented as illustrated in Figure 2.11.

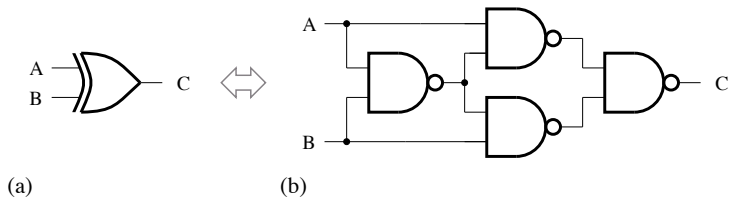


Figure 2.11. *XOR gate: a) symbol; b) construction using NAND gates*

For the XNOR gate, we have:

$$C = \overline{A \oplus B} = \overline{\overline{A} \cdot B + A \cdot \overline{B}} \qquad [2.9]$$

Because

$$\overline{A} \cdot B = \overline{\overline{\overline{A} \cdot B}} = \overline{\overline{\overline{A}(A + B)}} = \overline{A + \overline{A + B}} \qquad [2.10]$$

and

$$A \cdot \overline{B} = \overline{\overline{A \cdot \overline{B}}} = \overline{\overline{(A + B)\overline{B}}} = \overline{\overline{A + \overline{B} + B}} \qquad [2.11]$$

equation [2.9] becomes:

$$C = \overline{\overline{\overline{A + \overline{A + B}} + \overline{A + \overline{A + B}} + B}} \qquad [2.12]$$

Figure 2.12 depicts the logic circuit corresponding to equation [2.12].

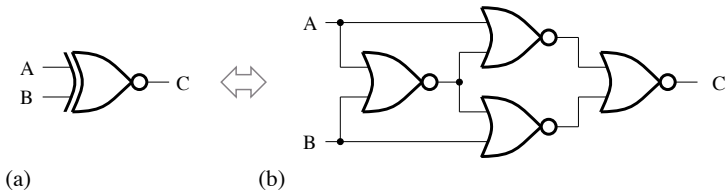


Figure 2.12. *XNOR gate: a) symbol; b) construction using NOR gates*

2.6.1. *Boolean algebra theorems*

Boolean algebra is governed by a certain number of theorems (or properties). The algebraic method of simplification uses properties of Boolean algebra to make it possible to minimize Boolean expressions (or logic functions), thus reducing the material cost for practical implementation.

2.6.1.1. *NOT, AND and OR functions*

Table 2.9 gives the basic properties for the NOT, AND and OR operations.

NOT	AND	OR
$\overline{0} = 1$	$0 \cdot X = 0$	$0 + X = X$
$\overline{1} = 0$	$1 \cdot X = X$	$1 + X = 1$
$\overline{\overline{X}} = X$	$X \cdot X = X$	$X + X = X$
	$X \cdot \overline{X} = 0$	$X + \overline{X} = 1$

Table 2.9. *Basic properties for the NOT, AND and OR operations*

In general, for the Boolean functions X, Y and Z, it is possible to establish the following theorems:

– *simplification theorem*:

$$X + X \cdot Y = X$$
$$X(X + Y) = X$$
$$X \cdot Y + X \cdot \overline{Y} = X$$
$$(X + Y)(X + \overline{Y}) = X$$

– *absorption theorem*:

$$X + \overline{X} \cdot Y = X + Y$$
$$X(\overline{X} + Y) = X \cdot Y$$

– *factorization and multiplication theorem*:

$$(X + Y)(\overline{X} + Z) = X \cdot Z + \overline{X} \cdot Y$$
$$X \cdot Y + \overline{X} \cdot Z = (X + Z)(\overline{X} + Y)$$

– *consensus theorem*:

$$X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z$$
$$(X + Y)(\overline{X} + Z)(Y + Z) = (X + Y)(\overline{X} + Z)$$

SHANNON'S EXPANSION THEOREM.– Let $F(x_0, x_1, \ldots, x_i, \ldots, x_{n-1})$ be a Boolean logic function of n variables. Shannon's expansion theorem can be written as follows:

$$F(x_0, x_1, \ldots, x_i, \ldots, x_{n-1}) = \overline{x_i} \cdot F(x_0, x_1, \ldots, 0, \ldots, x_{n-1}) + x_i \cdot F(x_0, x_1, \ldots, 1, \ldots, x_{n-1}) \quad [2.13]$$

In practice, applying Shannon's expansion theorem makes it possible to decompose a function of five variables, for example, to give rise to two functions of four variables. Thus:

$$F(A, B, C, D, E) = \overline{E} \cdot F(A, B, C, D, 0) + E \cdot F(A, B, C, D, 1) \quad [2.14]$$

where in fact $F(A, B, C, D, 0)$ and $F(A, B, C, D, 1)$ represent functions of four variables.

By iteratively applying Shannon's expansion theorem, a Boolean function can be expressed from the different values obtained for the different combinations of input variables as given in the truth table.

A	B	F(A,B)
0	0	1
0	1	0
1	0	0
1	1	1

Table 2.10. *Truth table*

For the two-variable function, $F(A, B)$, with the truth table in Table 2.10, we have:

$$\begin{aligned} F(A,B) &= \overline{A} \cdot F(0,B) + A \cdot F(1,B) \\ &= \overline{A}[\overline{B} \cdot F(0,0) + B \cdot F(0,1)] + A[\overline{B} \cdot F(1,0) + B \cdot F(1,1)] \\ &= \overline{A} \cdot \overline{B} \cdot F(0,0) + \overline{A} \cdot B \cdot F(0,1) + A \cdot \overline{B} \cdot F(1,0) + A \cdot B \cdot F(1,1) \end{aligned} \quad [2.15]$$

where $F(0,0) = 1$, $F(0,1) = 0$, $F(1,0) = 0$ and $F(1,1) = 1$.

2.6.1.2. *XOR (exclusive OR) and XNOR (inclusive AND) functions*

The XOR (OR exclusive) function of two variables is defined by:

$$X \oplus Y = X\overline{Y} + \overline{X} \cdot Y \quad [2.16]$$

Furthermore, we can write:

$$X \oplus Y = \overline{X} \oplus \overline{Y} \quad [2.17]$$

The definition of the XNOR (AND inclusive) function with two variables is given by:

$$X \odot Y = X \cdot Y + \overline{X} \cdot \overline{Y} \quad [2.18]$$

It must be noted that:

$$X \odot Y = \overline{X} \odot \overline{Y} = \overline{X \oplus Y} = \overline{X} \oplus Y = X \oplus \overline{Y} \quad [2.19]$$

Let us consider the Boolean functions, X, Y, and Z. Table 2.11 shows the truth table for the function $X \oplus (Y \cdot Z)$ and $(X \oplus Y)(X \oplus Z)$. We can thus observe that $X \oplus (Y \cdot Z)$ is different from $(X \oplus Y)(X \oplus Z)$.

X	Y	Z	$Y \cdot Z$	$X \oplus Y$	$X \oplus Z$	$X \oplus (Y \cdot Z)$	$(X \oplus Y)(X \oplus Z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	0	1	0
1	1	0	0	0	1	1	0
1	1	1	1	0	0	0	0

Table 2.11. *Truth table for $X \oplus (Y \cdot Z)$ and $(X \oplus Y)(X \oplus Z)$*

For the functions $X \oplus Y \oplus (X + Y)$ and $X \oplus Y \oplus X + X \oplus Y \oplus Y$, the truth table is represented in Table 2.12. Expressing all combinations of the variables X and Y where each function takes the logic level 1, we have:

$$X \oplus Y \oplus (X + Y) = X \cdot Y \qquad [2.20]$$

and

$$\begin{aligned} X \oplus Y \oplus X + X \oplus Y \oplus Y &= X \cdot Y + X \cdot \overline{Y} + \overline{X} \cdot Y \\ &= X(Y + \overline{Y}) + \overline{X} \cdot Y \\ &= X + \overline{X} \cdot Y \\ &= X + Y \end{aligned} \qquad [2.21]$$

X	Y	$X + Y$	$X \oplus Y$	$X \oplus Y \oplus (X + Y)$	$X \oplus Y \oplus X + X \oplus Y \oplus Y$
0	0	0	0	0	0
0	1	1	1	0	1
1	0	1	1	0	1
1	1	1	0	1	1

Table 2.12. *Truth table for $X \oplus Y \oplus (X + Y)$ and $X \oplus Y \oplus X + X \oplus Y \oplus Y$*

Therefore, functions $X \oplus Y \oplus (X + Y)$ and $X \oplus Y \oplus X + X \oplus Y \oplus Y$ are not equal.

The basic properties for the XOR and XNOR operations are given in Table 2.13.

XOR	XNOR
$0 \oplus X = X$	$0 \odot X = \overline{X}$
$1 \oplus X = \overline{X}$	$1 \odot X = X$
$X \oplus X = 0$	$X \odot X = 1$
$X \oplus \overline{X} = 1$	$X \odot \overline{X} = 0$

Table 2.13. *Basic properties for the XOR and XNOR operations*

The following theorems are verified for the Boolean functions X, Y and Z:

– *commutativity*:

$$X \oplus Y = Y \oplus X$$
$$X \odot Y = Y \odot X$$

– *associativity*:

$$X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z = X \oplus Y \oplus Z$$
$$X \odot (Y \odot Z) = (X \odot Y) \odot Z = X \odot Y \odot Z$$

– *factorization and distributivity*:

$$(X \cdot Y) \oplus (X \cdot Z) = X \cdot (Y \oplus Z)$$
$$(X + Y) \odot (X + Z) = X + (Y \odot Z)$$

– *absorption*:

$$X \cdot (\overline{X} \oplus Y) = X \cdot Y$$
$$X + (\overline{X} \odot Y) = X + Y$$

– *consensus*:

$$(X \cdot Y) \oplus (\overline{X} \cdot Z) + Y \cdot Z = (X \cdot Y) \oplus (\overline{X} \cdot Z)$$
$$(X + Y) \odot (\overline{X} + Z) \cdot (Y + Z) = (X + Y) \odot (\overline{X} + Z)$$

For two logic functions, X and Y, it may be established that:

– if $X \cdot Y = 0$, then $X + Y = X \oplus Y$;

– if $X + Y = 1$, then $X \cdot Y = X \odot Y$.

NOTE 2.1.– Boolean algebra theorems (or properties) possess two forms, one deduced from the other by replacing:

– all plus signs (+) with a point ($\cdot$), and vice versa;

– all circled plus signs ($\oplus$) with a circled point ($\odot$), and vice versa;

– any logic level 1 by logic level 0, and vice versa.

EXAMPLE 2.2.– Show that:

$$(X \cdot Y) \oplus (X + Y) = X \oplus Y$$
$$X \odot Y \odot (X \cdot Y) = X + Y$$
$$(\overline{X} + Y) \odot (X \oplus Y) = \overline{X} \cdot Y$$
$$X \cdot Y + Y \cdot Z + X \cdot Z = X \cdot Y \oplus Y \cdot Z \oplus X \cdot Z$$

Using Boolean algebra theorems (or properties), we can write:

$$\begin{aligned}(X \cdot Y) \oplus (X + Y) &= X \cdot Y(\overline{X + Y}) + \overline{X \cdot Y}(X + Y)\\ &= X \cdot Y(\overline{X + Y}) + \overline{X \cdot Y}(X + Y)\\ &= X \cdot Y(\overline{X} \cdot \overline{Y}) + (\overline{X} + \overline{Y})(X + Y)\\ &= X \cdot \overline{Y} + \overline{X} \cdot Y\\ &= X \oplus Y\end{aligned} \quad [2.22]$$

$$\begin{aligned}X \odot Y \odot (X \cdot Y) &= (X \odot Y)(X \cdot Y) + (\overline{X \odot Y})(\overline{X \cdot Y})\\ &= (X \cdot Y + \overline{X} \cdot \overline{Y})(X \cdot Y) + (X \oplus Y)(\overline{X} + \overline{Y})\\ &= X \cdot Y + (X \cdot \overline{Y} + \overline{X} \cdot Y)(\overline{X} + \overline{Y})\\ &= X \cdot Y + X \cdot \overline{Y} + \overline{X} \cdot Y\\ &= X(Y + \overline{Y}) + \overline{X} \cdot Y\\ &= X + \overline{X} \cdot Y\\ &= X + Y\end{aligned} \quad [2.23]$$

$$\begin{aligned}(\overline{X} + Y) \odot (X \oplus Y) &= (\overline{X} + Y) \odot (X \cdot \overline{Y} + \overline{X} \cdot Y)\\ &= (\overline{X} + Y) \cdot (X \cdot \overline{Y} + \overline{X} \cdot Y)\\ &\quad \text{because} \quad (\overline{X} + Y) + (X \cdot \overline{Y} + \overline{X} \cdot Y) = 1\\ &= \overline{X} \cdot Y\end{aligned} \quad [2.24]$$

$$
\begin{aligned}
X \cdot Y + Y \cdot Z + X \cdot Z &= X \cdot Y(Z + \overline{Z}) + (X + \overline{X})Y \cdot Z + X \cdot (Y + \overline{Y})Z \\
&= X \cdot Y \cdot Z + X \cdot Y \cdot \overline{Z} + \overline{X} \cdot Y \cdot Z + X \cdot \overline{Y} \cdot Z \\
&= X \cdot Y \cdot Z + X \cdot Y \cdot \overline{Z} \oplus (\overline{X} \cdot Y \cdot Z + X \cdot \overline{Y} \cdot Z) \\
&\qquad \text{because} \quad (X \cdot Y \cdot \overline{Z})(\overline{X} \cdot Y \cdot Z + X \cdot \overline{Y} \cdot Z) = 0 \\
&= X \cdot Y(Z + \overline{Z}) \oplus (Y \oplus X)Z \\
&= X \cdot Y \oplus Y \cdot Z \oplus X \cdot Z
\end{aligned}
\qquad [2.25]
$$

2.6.2. *Karnaugh maps*

A semi-graphical method, which is based on the use of Karnaugh maps, is more appropriate for the simplification of more complex Boolean expressions.

A Karnaugh map, like a truth table, provides a representation of logic functions. It is composed of a certain number of squares or cells, each of which is reserved for a term (minterm or maxterm) of a logic function. Figure 2.13 shows Karnaugh maps for a three-variable function and Figure 2.14 presents Karnaugh maps for a four-variable function. The variables can be represented in two ways. On each map, the combination of the variables are placed in accordance with the order of Gray's encoding such that adjacent terms are in the neighboring cells or in the cells at map ends.

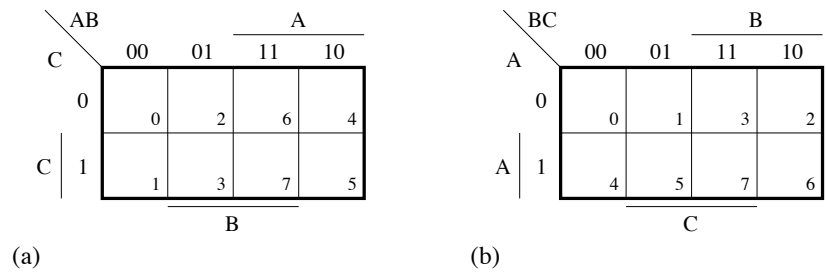


Figure 2.13. *Three-variable Karnaugh map*

Though Karnaugh maps can be used to reduce any logic function, they become difficult to manipulate when the number of variables exceeds six.

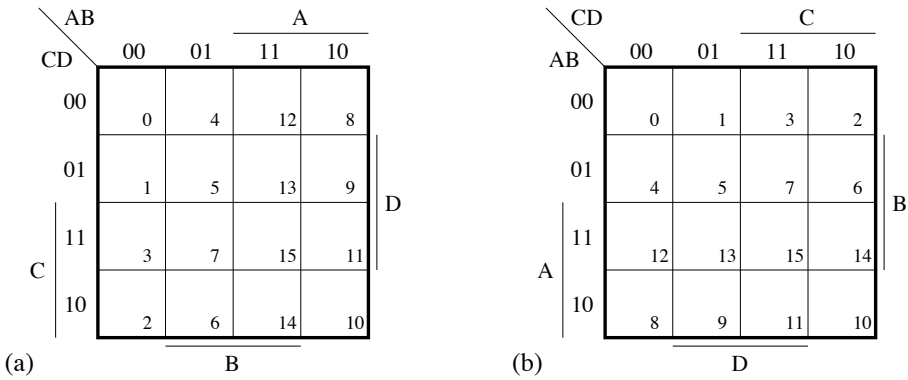


Figure 2.14. *Four-variable Karnaugh map*

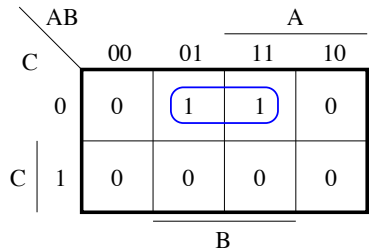


Figure 2.15. *Duad:*
$X = \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot \overline{C} = B \cdot \overline{C}$

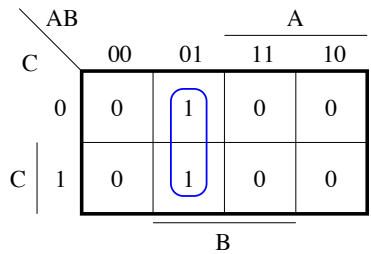


Figure 2.16. *Duad:*
$X = \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot B \cdot C = \overline{A} \cdot B$

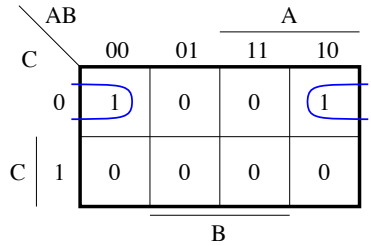


Figure 2.17. *Duad:*

$$X = \overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} = \overline{B} \cdot \overline{C}$$

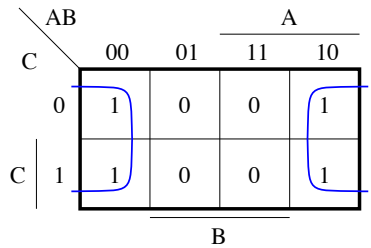


Figure 2.18. *Quad:*

$$X = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C = \overline{B}$$

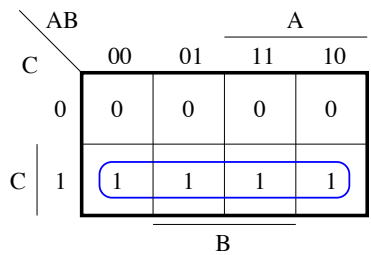


Figure 2.19. *Quad:*

$$X = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C + A \cdot B \cdot C + A \cdot \overline{B} \cdot C \ = C$$

Using a Karnaugh map, the simplification of a logic function is carried out by grouping the adjacent cells that contain 1s. The number of cells in a group must be a power of 2, or of the form 2^n ($n = 1, 2, 3, \ldots$):

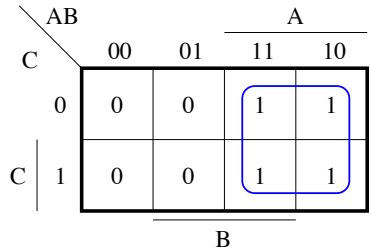


Figure 2.20. *Quad:*

$$X = A \cdot B \cdot C + A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot \overline{C} = A$$

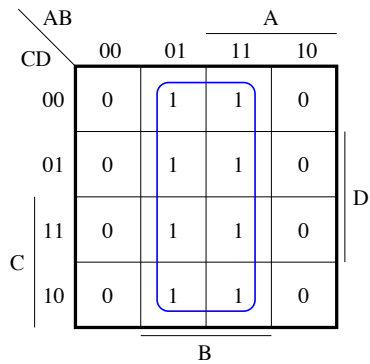


Figure 2.21. *Octad:*

$$X = B$$

– duad of adjacent 1s: the variable that is both complemented and non-complemented can be eliminated (see Figures 2.15–2.17);

– quad of adjacent 1s: two variables that are both complemented and non-complemented can be eliminated (see Figures 2.18–2.20);

– octad of adjacent 1s: three variables that are both complemented and non-complemented can be eliminated (see Figures 2.21–2.24).

Only those variables that hold the same logic state in all the cells of a group appear in the simplified expression.

NOTE 2.2.– In the case of several possibilities for grouping the cells of a Karnaugh map, the minimization procedure can yield more than one logic expression.

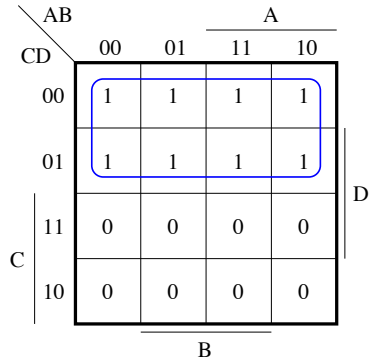


Figure 2.22. *Octad:*

$X = \overline{C}$

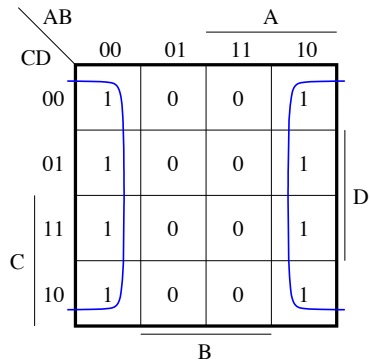


Figure 2.23. *Octad:*

$X = \overline{B}$

The *minterms* that are not necessary for the desired application may be used to improve the simplification of the logic circuit. They are considered as *don't care terms*.

EXAMPLE 2.3.– Simplify the following expressions:

1) $X(A, B, C) = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{B} \cdot C + \overline{A} \cdot B$.

2) $X(A, B, C) = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C}$. We shall suppose that the input condition $A \cdot \overline{B} \cdot C$ does not affect the logic level of the function X (*do not care condition*).

3) $X(A,B,C,D) = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D + \overline{A}\cdot B\cdot\overline{C}\cdot D + \overline{A}\cdot B\cdot C\cdot D + \overline{A}\cdot B\cdot C\cdot\overline{D}$
$+A\cdot B\cdot C\cdot\overline{D} + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D} + A\cdot\overline{B}\cdot\overline{C}\cdot D + A\cdot\overline{B}\cdot C\cdot\overline{D}.$

4) $X(A,B,C,D) = \overline{A}\cdot\overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot\overline{C}\cdot\overline{D} + \overline{A}\cdot B\cdot\overline{C}\cdot D + \overline{A}\cdot B\cdot C\cdot D$
$+\overline{A}\cdot B\cdot C\cdot\overline{D} + A\cdot B\cdot\overline{C}\cdot\overline{D} + A\cdot B\cdot\overline{C}\cdot D$

5) $X(A,B,C,D) = \overline{A}\cdot\overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot\overline{C}\cdot D + \overline{A}\cdot B\cdot C\cdot D + \overline{A}\cdot B\cdot C\cdot\overline{D}$
$+A\cdot B\cdot\overline{C}\cdot\overline{D} + A\cdot B\cdot\overline{C}\cdot D + A\cdot B\cdot C\cdot D + A\cdot\overline{B}\cdot C\cdot D$

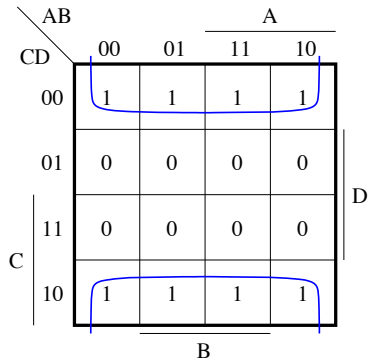


Figure 2.24. *Octad:*

$X = \overline{D}$

Using Karnaugh maps represented in Figures 2.25–2.30, different solutions can be obtained.

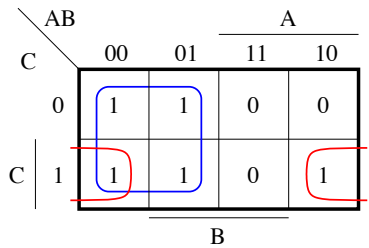


Figure 2.25. *Example 2.1(1):*

$X = \overline{A} + \overline{B}\cdot C$

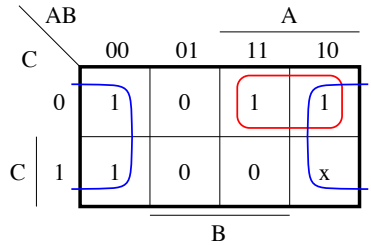


Figure 2.26. *Example 2.1(2):*

$$X = \overline{B} + A \cdot \overline{C}$$

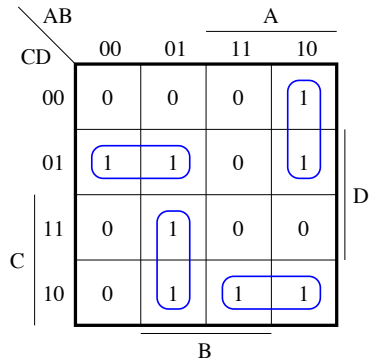


Figure 2.27. *Example 2.1(3) (case 1):*

$$X = \overline{A} \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot C \cdot \overline{D}$$

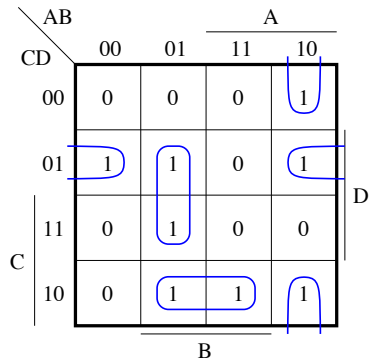


Figure 2.28. *Example 2.1(3) (case 2):*

$$X = \overline{A} \cdot B \cdot D + B \cdot C \cdot \overline{D} + \overline{B} \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot \overline{D}$$

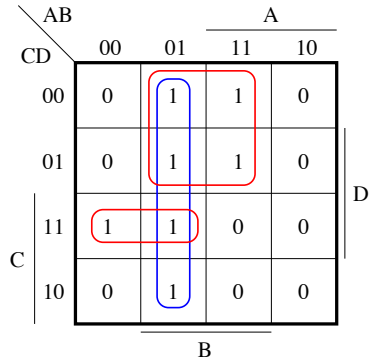


Figure 2.29. *Example 2.1(4):*

$X = \overline{A} \cdot B + B \cdot \overline{C} + \overline{A} \cdot C \cdot D$

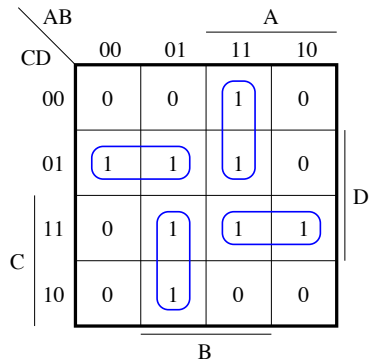


Figure 2.30. *Example 2.1(5):*

$X = A \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C + A \cdot C \cdot D$

It should be noted that some logic functions can have many minimal expressions.

EXAMPLE 2.4.– Simplify the logic function F in the two following cases:

a) $F(A, B, C) = \sum m(1, 3, 4, 7)$;

b) $F(A, B, C) = \sum m(1, 3, 4, 7) + x(2, 5)$, where the *don't care terms* are represented by x.

The minimal expressions of the function F may be obtained from the Karnaugh maps represented in Figures 2.31 and 2.32.

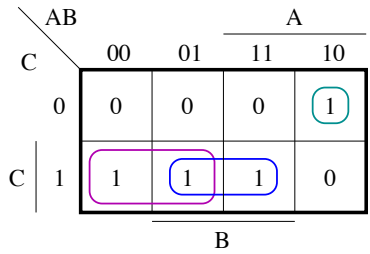


Figure 2.31. *Example 2.2(a):*
$F = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot C + B \cdot C$

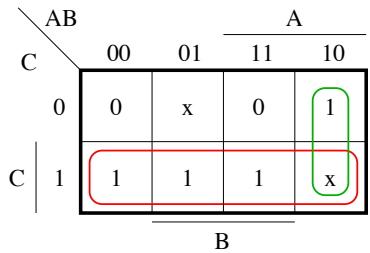


Figure 2.32. *Example 2.2(b):*
$F = A \cdot \overline{B} + C$

It should be noted that a *don't care term* is taken into account only if it can contribute to the simplification of the logic function.

2.6.3. *Simplification of logic functions with multiple outputs*

The simplification of logic functions with multiple outputs can be carried out in four steps:

1) Write the functions to be simplified in the sum of products;

2) The minterms being represented by m and the *don't care terms* by x, form the products of sums in a systematic manner in accordance with the following rules of the AND operation:

$$m_i \cdot m_i = m_i$$
$$m_i \cdot m_j = 0 \ (i \neq j)$$
$$m_i \cdot \mathrm{x}_i = m_i$$
$$\mathrm{x}_i \cdot \mathrm{x}_i = \mathrm{x}_i$$
$$m_i \cdot \mathrm{x}_j = \mathrm{x}_i \cdot \mathrm{x}_j = 0 \ (i \neq j);$$

3) Draw up a table containing the terms common to different functions;

4) Based on the Karnaugh map for each function, group the common terms and then simplify the remaining terms.

However, when a group of common terms is part of a larger group of 2^n $(n = 1, 2, 3)$ terms, this group is only selected if it yields the simplest logic expression.

EXAMPLE 2.5.– Propose a circuit that implements the following circuit:

$$F(A, B, C, D) = \sum m(0, 2, 3, 4, 6, 7, 10, 11)$$
$$G(A, B, C, D) = \sum m(0, 4, 8, 9, 10, 11, 12, 13)$$

Taking into account the common terms (see the Karnaugh maps shown in Figures 2.33 and 2.34), we can obtain the circuit in Figure 2.35 that consists of six logic gates with a total of 16 inputs.

By independently simplifying the two functions, we arrive at:

$$F(A, B, C, D) = \overline{A} \cdot C + \overline{A} \cdot \overline{D} + \overline{B} \cdot C$$
$$G(A, B, C, D) = A \cdot \overline{C} + \overline{C} \cdot \overline{D} + A \cdot \overline{B}$$

In this case, eight logic gates with a total of 18 inputs are needed for the implementation of the functions F and G.

2.6.4. *Factorization of logic functions*

To reduce hardware implementation costs, it is often necessary to find terms common to several functions.

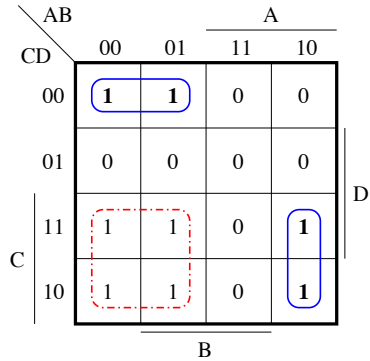


Figure 2.33. *Function F*
$F = \overline{A} \cdot C + A \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{C} \cdot \overline{D}$

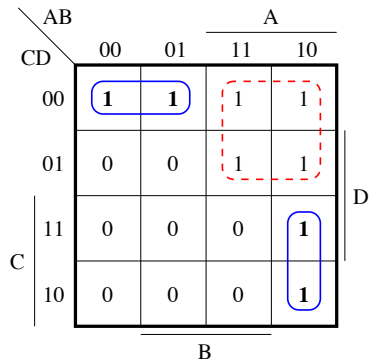


Figure 2.34. *Function G*
$G = A \cdot \overline{C} + A \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{C} \cdot \overline{D}$

Factorize (or decompose) the following Boolean expressions to yield the term $C + D$:

$$F(A, B, C, D) = A \cdot C + A \cdot D + B \cdot \overline{C} \cdot \overline{D}$$

$$G(A, B, C, D) = A \cdot B \cdot C + A \cdot B \cdot D + \overline{A} \cdot \overline{C} \cdot \overline{D} + \overline{B} \cdot \overline{C} \cdot \overline{D}$$

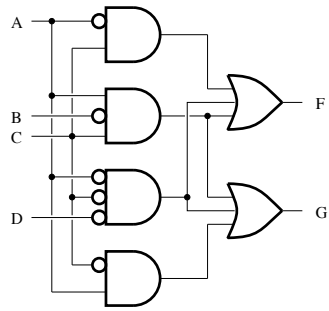


Figure 2.35. *Logic circuits for the implementation of F and G*

The factorization (or the decomposition) of functions F and G can be carried out as follows:

$$F = A \cdot C + A \cdot D + B \cdot \overline{C} \cdot \overline{D} \quad [2.26]$$

$$= A(C + D) + B \cdot \overline{C} \cdot \overline{D}$$

$$= A(C + D) + B(\overline{C + D}) \quad [2.27]$$

and

$$G = A \cdot B \cdot C + A \cdot B \cdot D + \overline{A} \cdot \overline{C} \cdot \overline{D} + \overline{B} \cdot \overline{C} \cdot \overline{D} \quad [2.28]$$

$$= A \cdot B(C + D) + \overline{A}(\overline{C + D}) + \overline{B}(\overline{C + D})$$

$$= A \cdot B(C + D) + \overline{AB}(\overline{C + D}) \quad [2.29]$$

2.7. Multi-level logic circuit implementation

Combinational logic circuits are generally designed using two-level logic networks. They arise directly from the sum-of-product expressions and are characterized by a high speed of operation (or fast response time) if the number of inputs is such that the load limit for each logic gate is not exceeded (or if the input fan-in specification is satisfied).

The design of multi-level circuits is based on the factorization and decomposition of logic functions which are taken in their minimal form. In practice, the use of supplementary levels in a circuit helps to reduce the maximum number of inputs for logic gates to the value permitted by the manufacturing technology.

Conventionally, input inverters are not considered when determining the number of circuit levels as they may be affected by the type of logic circuit (active high or active low) to be chosen.

2.7.1. *Examples*

The five-variable function, $F(A, B, C, D, E)$, is to be implemented. It is expressed in the sum-of-product form as:

$$F(A, B, C, D, E) = A \cdot C + A \cdot D + B \cdot C + B \cdot D + \overline{E} \qquad [2.30]$$

This function can be implemented using four 2-input AND gates or one 5-input OR gate as shown in Figure 2.36(a).

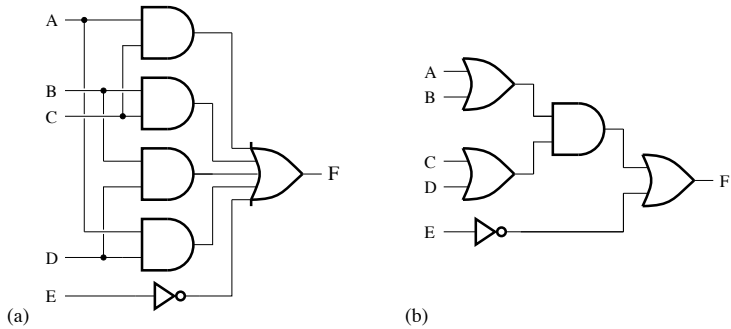


Figure 2.36. *Logic circuit for the implementation of F*

Upon factorizing, we can also arrive at:

$$F(A, B, C, D, E) = (A + B)(C + D) + \overline{E} \qquad [2.31]$$

As a result, the function $F(A, B, C, D, E)$ can be implemented using three 2-input OR gates and one 2-input AND gate, as illustrated in Figure 2.36(b). This approach, in addition to reducing the number of logic gates, makes it possible to improve the *fan-in* of logic gates.

In the case of a logic function of four variables, $G(A, B, C, D)$, given in the sum-of-products form by:

$$G(A, B, C, D) = \overline{A} \cdot C + \overline{A} \cdot D + \overline{B} \cdot C + \overline{B} \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D} \qquad [2.32]$$

the implementation, as illustrated in Figure 2.37(a), requires four 2-input AND gates, one 4-input AND gate and one 5-input OR gate.

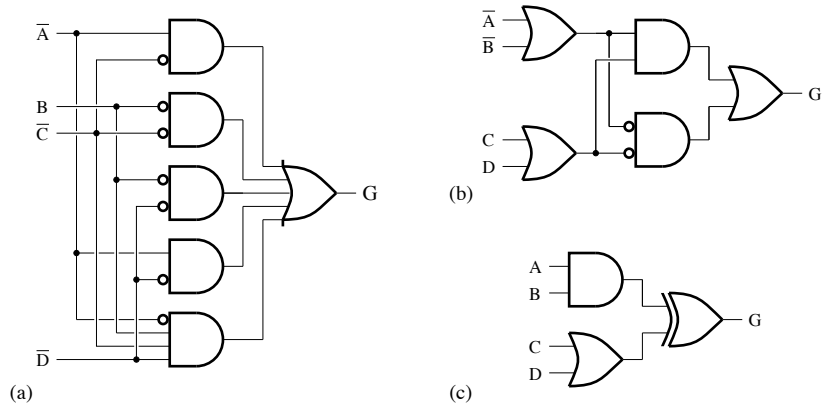


Figure 2.37. *Logic circuit for the implementation of G*

The implementation as shown in Figure 2.37(b) is based on the factorization of the function $G(A, B, C, D)$ as follows:

$$\begin{aligned} G(A, B, C, D) &= (\overline{A} + \overline{B})(C + D) + (\overline{\overline{A} + \overline{B}})(\overline{C + D}) \\ &= (\overline{A \cdot B})(C + D) + (A \cdot B)(\overline{C + D}) \end{aligned} \quad [2.33]$$

It has three levels and uses only 2-input logic gates (three OR gates and two AND gates).

We observe that the function $G(A, B, C, D)$ can also be written in the following manner:

$$G(A, B, C, D) = A \cdot B \oplus (C + D) \quad [2.34]$$

which yields another logic circuit, as shown in Figure 2.37(c), and which is composed of one OR gate, one AND gate and one XOR gate.

2.7.2. *NAND gate logic circuit*

Based on DeMorgan's theorems, a logic circuit consisting of AND and OR gates can be transformed to a circuit made up solely of NAND gates (1) by replacing the AND gates with NAND gates, adding inverters at the OR gate inputs and by inserting inverters wherever necessary to correct for the effect of non-compensated inversions,

and, lastly, (2) by replacing all OR gates with input inverters to NAND gates (see Figure 2.38).

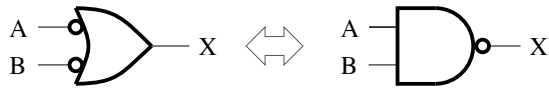


Figure 2.38. *Equivalent circuits for the NAND gate*

Consider the following logic function of four variables A, B, C and D:

$$F(A, B, C, D) = (A + \overline{B})(C \cdot \overline{D} + \overline{C} \cdot D) \qquad [2.35]$$

This function may be implemented by using AND and OR gates, as illustrated in Figure 2.39(a). By applying the transformations (1) and (2), we can obtain the circuit shown in Figure 2.39(b) and the NAND gate based circuit shown in Figure 2.39(c), respectively.

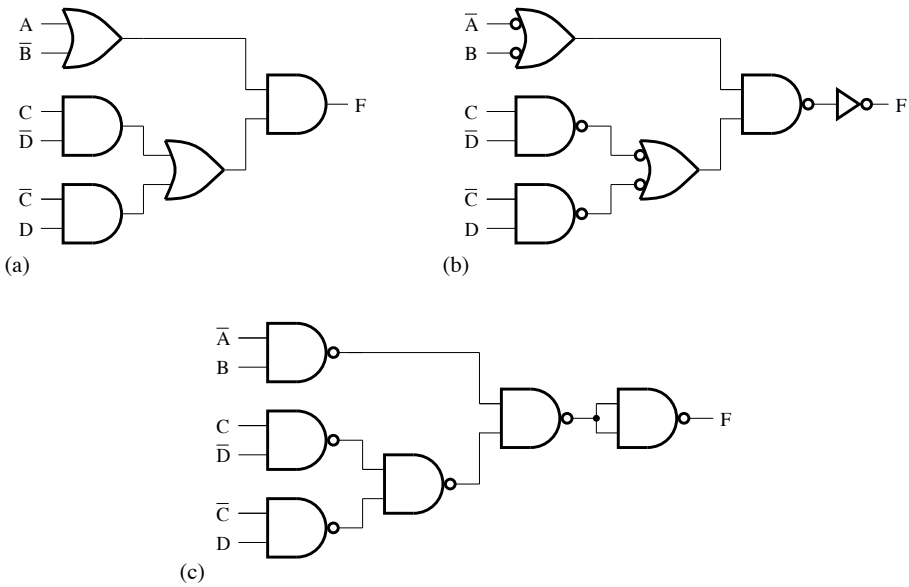


Figure 2.39. *Implementation of the function F: a) circuit using AND and OR gates; b) equivalent circuit; c) circuit based on NAND gates*

Let us consider another function G of four variables A, B, C and D, defined by:

$$G = A \cdot B \cdot \overline{C} + A \cdot \overline{D} + \overline{B} \cdot \overline{D} + \overline{C} \cdot \overline{D} \qquad [2.36]$$

To implement this function using only 2-input NAND gates, we first observe that:

$$G = (A + \overline{B \cdot \overline{C}})(B \cdot \overline{C} + \overline{D}) \qquad [2.37]$$

and subsequently derive the circuit built up of AND and OR gates, as illustrated in Figure 2.40(a). Using transformations based on DeMorgan's theorems, we can obtain the equivalent circuit in Figure 2.40(b), and then the NAND gate based circuit represented in Figure 2.40(c).

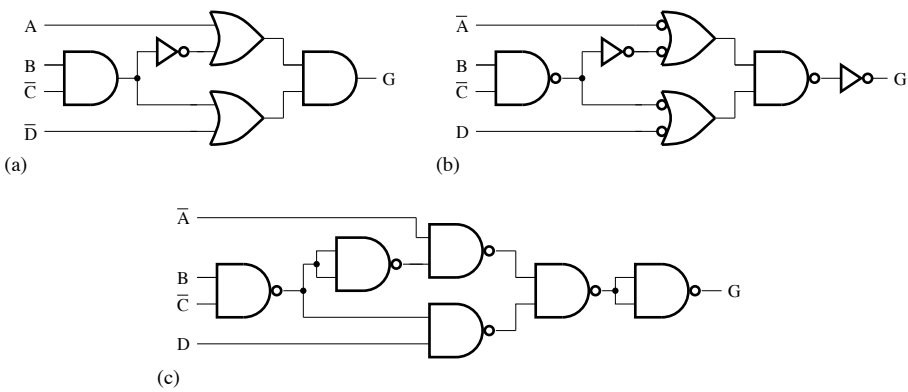


Figure 2.40. *Implementation of the function G: a) circuit using AND and OR gates; b) equivalent circuit; c) NAND gate based circuit*

NOTE 2.3.– Each sum-of-products logic expression corresponds to a circuit using AND and OR gates, or a NAND gate based circuit.

2.7.3. *NOR gate based logic circuit*

A logic circuit built up of AND and OR logic gates can be transformed in accordance with DeMorgan's theorems to a circuit consisting solely of NOR gates: (1) by replacing the OR gates by NOR gates; by adding input inverters to the AND gates and by inserting inverters wherever necessary to correct for the effect of non-compensated inversions; (2) by replacing all AND gates that have input inverters with NOR gates (see Figure 2.41).

Figure 2.41. *Equivalent circuits for the NOR gate*

Consider the following logic function with four variables:

$$F(A,B,C,D) = (A + \overline{B})(C \cdot \overline{D} + \overline{C} \cdot D) \qquad [2.38]$$

The implementation of this function using AND and OR gates is illustrated in Figure 2.42(a). For the NOR gate based implementation, the first step of the required transformation results in the circuit shown in Figure 2.42(b) that is then converted to the NOR gate based circuit as shown in Figure 2.42(c).

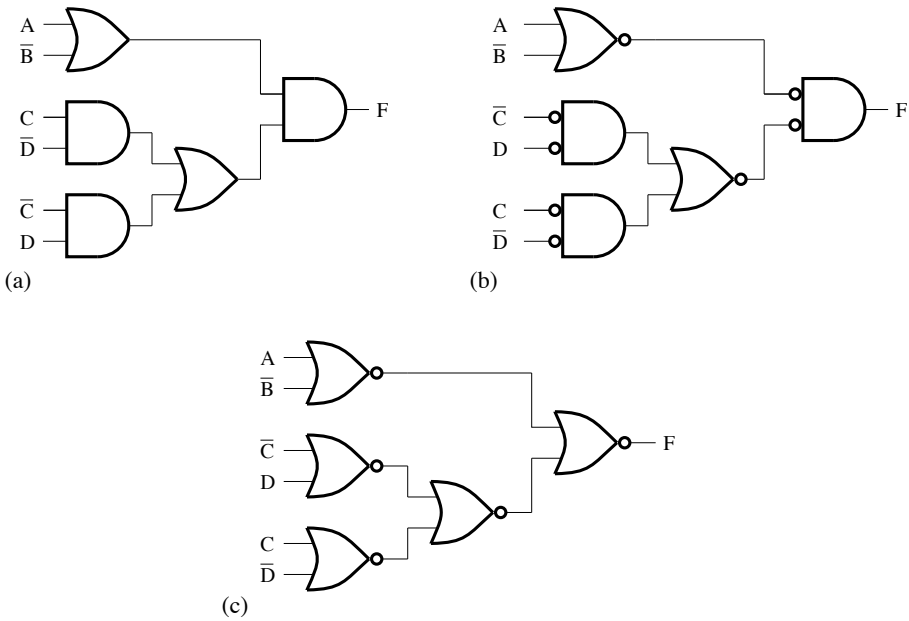


Figure 2.42. *Implementation of the function F: a) circuit using AND and OR gates; b) equivalent circuit; c) NOR gate based circuit*

Use only two-input NOR gates to implement the following function of four variables:

$$G = A \cdot B \cdot \overline{C} + A \cdot \overline{D} + \overline{B} \cdot \overline{D} + \overline{C} \cdot \overline{D} \quad [2.39]$$

Because:

$$G = (A + \overline{B \cdot \overline{C}})(B \cdot \overline{C} + \overline{D}) \quad [2.40]$$

we can derive the circuit shown in Figure 2.43(a) or the equivalent circuit shown in Figure 2.43(b). The NOR-gate based circuit corresponding to the function G is illustrated in Figure 2.43(c).

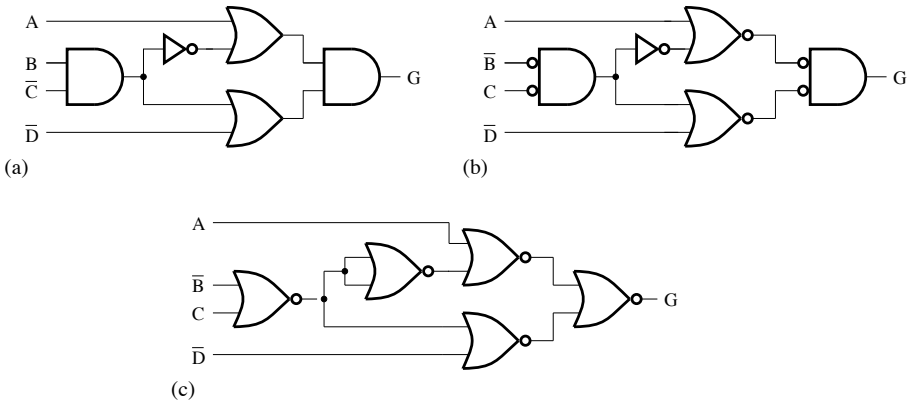


Figure 2.43. *Implementation of the function G: a) circuit using AND and OR gates; b) equivalent circuits; c) NOR gate based circuit*

NOTE 2.4.– Each product-of-sums logic expression corresponds to a circuit consisting of OR and AND gates or to a NOR gate based circuit.

2.7.4. *Representation based on XOR and AND operators*

The logic circuit for certain functions may be difficult to optimize when they are represented in the sum-of-products form. Using a representation based on XOR (or XNOR) gates often offers the advantage of making it possible to reduce the number of logic gates and the complexity of interconnection lines.

In addition to being a simplification method for logic functions, Karnaugh maps are also useful in identifying terms that can be more easily implemented using XOR

logic gates. In a Karnaugh map, groups of terms, that are symmetrical, or mirror images or duplicates of each other may be represented in XOR (or Reed–Muller) form. The following figures give some examples.

For the three-variable functions whose Karnaugh maps are represented in Figures 2.44 and 2.45, we have, respectively:

$$F_a = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} \quad [2.41]$$

$$= \overline{A}(B \oplus C) + A(\overline{B \oplus C})$$

$$= A \oplus B \oplus C \quad [2.42]$$

and

$$F_b = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot C + A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C \quad [2.43]$$

$$= \overline{A}(\overline{B \oplus C}) + A(B \oplus C)$$

$$= \overline{A \oplus B \oplus C} \quad [2.44]$$

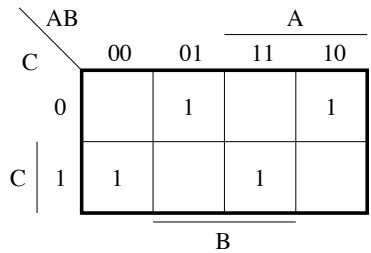


Figure 2.44. *Representation of* $F_a = A \oplus B \oplus C$

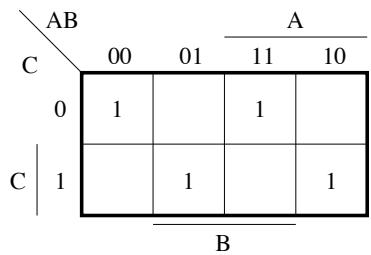


Figure 2.45. *Representation of* $F_b = \overline{A \oplus B \oplus C}$

In the case of the four-variable functions, defined by the Karnaugh maps shown in Figures 2.46–2.49, we have, respectively:

$$F_c = \overline{A} \cdot C + A \cdot \overline{C} = A \oplus C \quad [2.45]$$

$$F_d = \overline{A} \cdot B + A \cdot \overline{B} = A \oplus B \quad [2.46]$$

$$F_e = \overline{B} \cdot D + B \cdot \overline{D} = B \oplus D \quad [2.47]$$

and

$$F_f = \overline{B} \cdot \overline{C} + B \cdot C = \overline{B \oplus C} \quad [2.48]$$

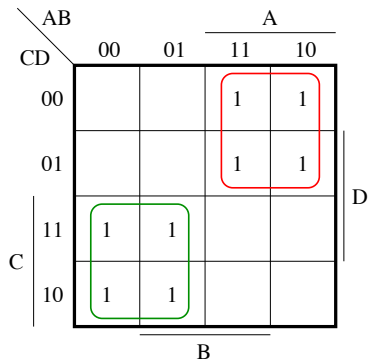


Figure 2.46. *Representation of* $F_c = A \oplus C$

Considering the Karnaugh map depicted in Figure 2.50, the equation for the logic function is written as follows:

$$F_g = \overline{A} \cdot C + C \cdot D + A \cdot \overline{C} \cdot \overline{D} \quad [2.49]$$

$$= C(\overline{A \cdot \overline{D}}) + \overline{C}(A \cdot \overline{D})$$

$$= C \oplus A \cdot \overline{D} \quad [2.50]$$

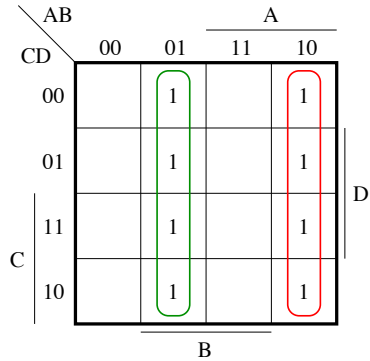


Figure 2.47. *Representation of* $F_d = A \oplus B$

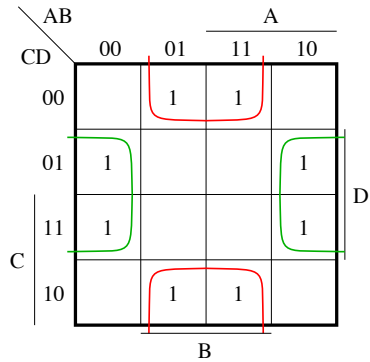


Figure 2.48. *Representation of* $F_e = B \oplus D$

We can determine the equation for the following logic function based on the Karnaugh map in Figure 2.51:

$$F_h = A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot C \cdot D + A \cdot B \cdot C \cdot \overline{D} \quad [2.51]$$

$$= A[(\overline{B} \cdot \overline{C} + B \cdot C)\overline{D} + (\overline{B} \cdot C + B \cdot \overline{C})D]$$

$$= A[(\overline{B \oplus C})\overline{D} + (B \oplus C)D]$$

$$= A(\overline{B \oplus C \oplus D}) \quad [2.52]$$

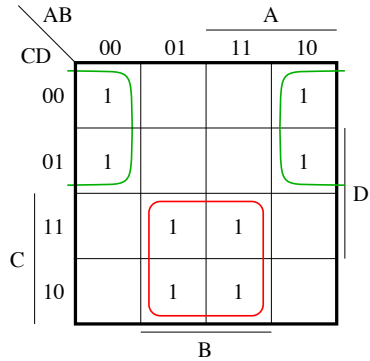


Figure 2.49. *Representation of* $F_f = \overline{B \oplus C}$

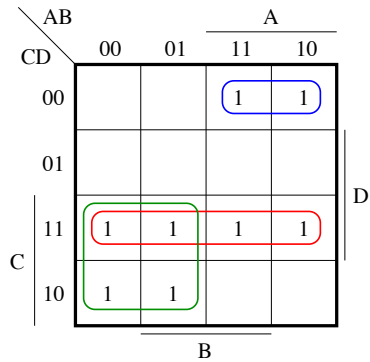


Figure 2.50. *Representation of* $F_g = C \oplus A \cdot \overline{D}$

For the Karnaugh map shown in Figure 2.52, the equation of the logic function can be obtained as follows:

$$F_i = \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot D \quad [2.53]$$

$$= (\overline{A} \cdot C + A \cdot \overline{C})(\overline{B} \cdot D + B \cdot \overline{D})$$

$$= (A \oplus C)(B \oplus D) \quad [2.54]$$

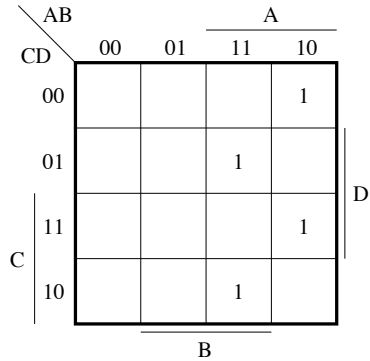


Figure 2.51. *Representation of* $F_h = A(\overline{B \oplus C \oplus D})$

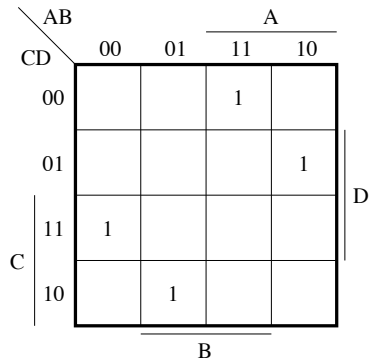


Figure 2.52. *Representation of* $F_i = (A \oplus C)(B \oplus D)$

The equation for the logic function represented by the Karnaugh map shown in Figure 2.53 is given by:

$$F_j = \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot \overline{D} \quad [2.55]$$

$$= (\overline{A} \cdot D + A \cdot \overline{D})(\overline{B} \cdot C + B \cdot \overline{C})$$

$$= (A \oplus D)(B \oplus C) \quad [2.56]$$

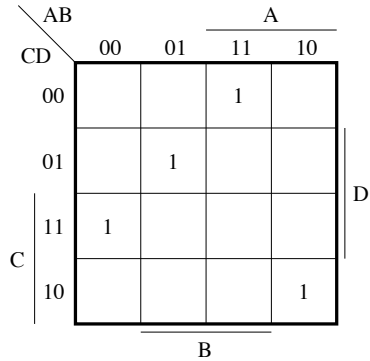


Figure 2.53. *Representation of* $F_j = (A \oplus D)(B \oplus C)$

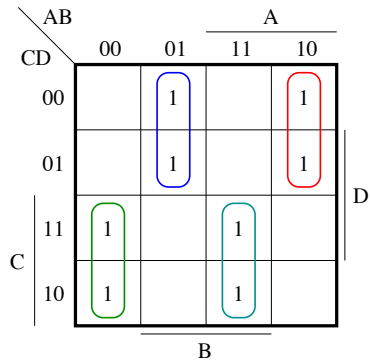


Figure 2.54. *Representation of* $F_k = A \oplus B \oplus C$

The Karnaugh map shown in Figure 2.54 corresponds to the logic function whose equation is written as follows:

$$F_k = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} \quad [2.57]$$

$$= \overline{A}(\overline{B} \cdot C + B \cdot \overline{C}) + A(\overline{B} \cdot \overline{C} + B \cdot C)$$

$$= \overline{A}(B \oplus C) + A(\overline{B \oplus C})$$

$$= A \oplus B \oplus C \quad [2.58]$$

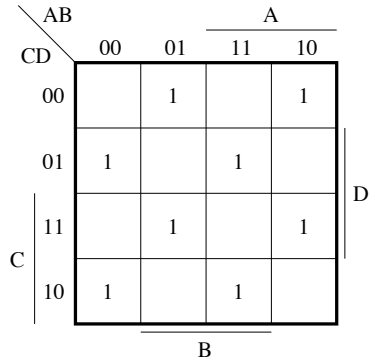


Figure 2.55. *Representation of* $F_l = A \oplus B \oplus C \oplus D$

Based on the Karnaugh map in Figure 2.54, we have:

$$F_l = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot D + A \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot C \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot D \quad [2.59]$$

$$= (\overline{A} \cdot \overline{B} + A \cdot B)(\overline{C} \cdot D + C \cdot \overline{D}) + (\overline{A} \cdot B + A \cdot \overline{B})(\overline{C} \cdot \overline{D} + C \cdot D)$$

$$= (\overline{A \oplus B})(C \oplus D) + (A \oplus B)(\overline{C \oplus D})$$

$$= A \oplus B \oplus C \oplus D \quad [2.60]$$

2.8. Practical considerations

In practice, the operation of a logic circuit depends on the electric characteristics of the logic gates.

In an ideal case, the logic levels 0 and 1 are represented by fixed voltages (for example ground and supply voltage) for both the input and the output. In reality, they correspond to voltages that can vary within a certain range of values.

When logic gates are combined in order to construct a logic circuit, we can connect two inputs together or one input to one output. But in no case can we connect two different outputs as they can produce different logic states.

A timing diagram is a graphical representation of how variables in a system evolve over time. In the case of logic circuits, time is represented on the horizontal axis and the logic levels (low or high) of the variables are represented on the vertical axis. In addition to some electric characteristics, the timing diagram provides information that

is useful in constructing a truth table. It also shows the rise time and the fall time for signals, and the propagation delay, that is the time that passes between a change in level of the input and the corresponding change in the output.

2.8.1. *Timing diagram for a logic circuit*

A timing diagram is a graphical representation of the temporal evolution of a logic signal.

Let us consider the logic circuit in Figure 2.56(a). The logic equation for the output signal is of the following form:

$$F = A \cdot \overline{C} + B \cdot C \qquad [2.61]$$

The timing diagram in an ideal case is illustrated in Figure 2.56(b). However, it may be affected by propagation delays that depend on the response times of the different logic gates.

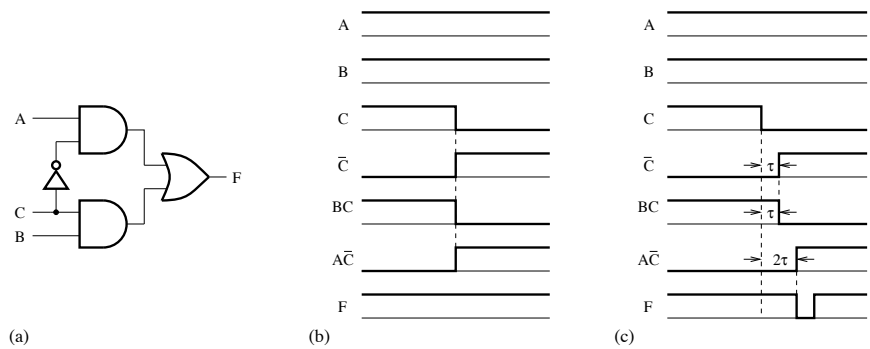


Figure 2.56. *a) Logic circuit; b) timing diagram in an ideal case; c) timing diagram illustrating the effect of a static hazard*

In general, a circuit that is sensitive to parasitic phenomena due to signal propagation along several paths may be affected by a hazard. There is a distinction made between static and dynamic hazards.

2.8.2. *Static hazard*

A static hazard is produced when a change in the level of an input variable, which should normally not bring about a modification of the output, translates into the generation of a transient signal with an erroneous logic level.

The logic circuit shown in Figure 2.56(a) contains two concurrent paths with different propagation delays. Thus, when the input signal C changes its logic level, the inputs $\overline{C}$ and B of one of the AND gates do not change simultaneously. This translates into a static hazard that can be seen in the timing diagram shown in Figure 2.56(c).

To suppress the effect of the static hazard on the operation of the two-level circuit represented in Figure 2.57(a), a product of terms must be introduced between the states $A\ B\ C = 1\ 1\ 1$ and $A\ B\ \overline{C} = 1\ 1\ 0$ (see Figure 2.57(b)). This helps to prevent the transition of the input F toward 0, as shown in Figure 2.57(c).

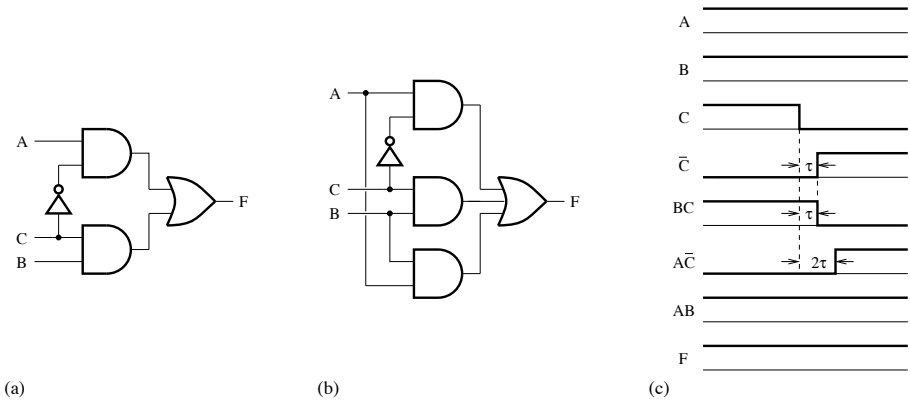


Figure 2.57. *a) Circuit with static hazard; b) logic circuit functioning without static hazard; c) timing diagram*

The minimal form of the logic equation for the output F is represented by the Karnaugh map in Figure 2.58, and the redundant term that must be added to eliminate the static hazard appears on the Karnaugh map in Figure 2.59.

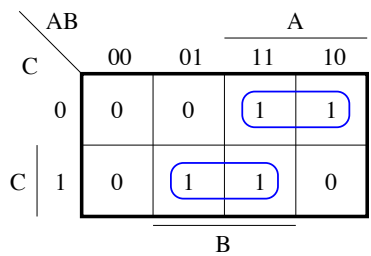


Figure 2.58. *Circuit with hazard*

$$F = A \cdot \overline{C} + B \cdot C$$

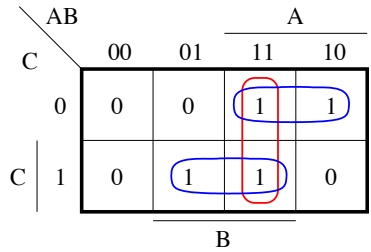


Figure 2.59. *Circuit without hazard*

$$F = A \cdot \overline{C} + B \cdot C + A \cdot B$$

In practice, the method used to eliminate static hazards that can affect the operation of a two-level logic circuit consists of identifying the terms corresponding to adjacent loops in the Karnaugh map and adding the appropriate redundant terms to the minimal representation of the output function.

2.8.3. *Dynamic hazard*

A multi-level logic circuit exhibits dynamic hazards if the transition of an input signal, which is supposed to produce a single change in the logic level of the output, changes the logic level of the output as desired only after a transient regime with at least two changes in logic level.

The logic circuit shown in Figure 2.60(a) is supposed to have two concurrent paths with asymmetrical propagation delays. The circuit operation may be affected by a dynamic hazard, as shown in Figure 2.60(b) for the output G. It must be noted that this circuit also presents a static hazard (output F). In this case, adding the logic gate that implements the redundant term $A \cdot B$ in order to eliminate the static hazard, as shown in Figure 2.61(a), also contributes in removing the effect of the dynamic hazard, as shown in Figure 2.61(b).

In general, a multi-level logic circuit cannot present static hazards but it may be affected by dynamic hazards. Therefore, for the circuit to operate without hazards, it is preferable to implement logic functions as two-level circuits and then to detect and eliminate static hazards by adding the appropriate redundant terms.

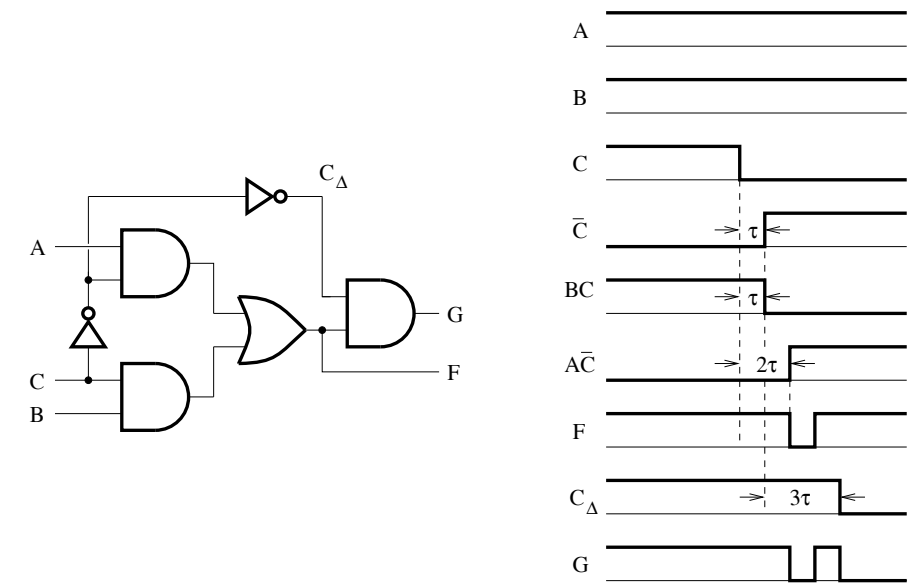


Figure 2.60. *a) Logic circuit and b) timing diagram illustrating the effect of a dynamic hazard*

2.9. Demonstration of some Boolean algebra identities

DEMONSTRATION 2.1.– Show that $X+X\cdot Y = X$. Using Boolean algebra theorems, we can write:

$$\begin{aligned} X + X \cdot Y &= X \cdot 1 + X \cdot Y \\ &= X(1+Y) \\ &= X(1) \\ &= X \end{aligned} \qquad [2.62]$$

DEMONSTRATION 2.2.– Show that $X(X+Y) = X$. In this case, we have:

$$\begin{aligned} X(X+Y) &= X \cdot X + X \cdot Y \\ &= X + X \cdot Y \\ &= X(1+Y) \\ &= X \end{aligned} \qquad [2.63]$$

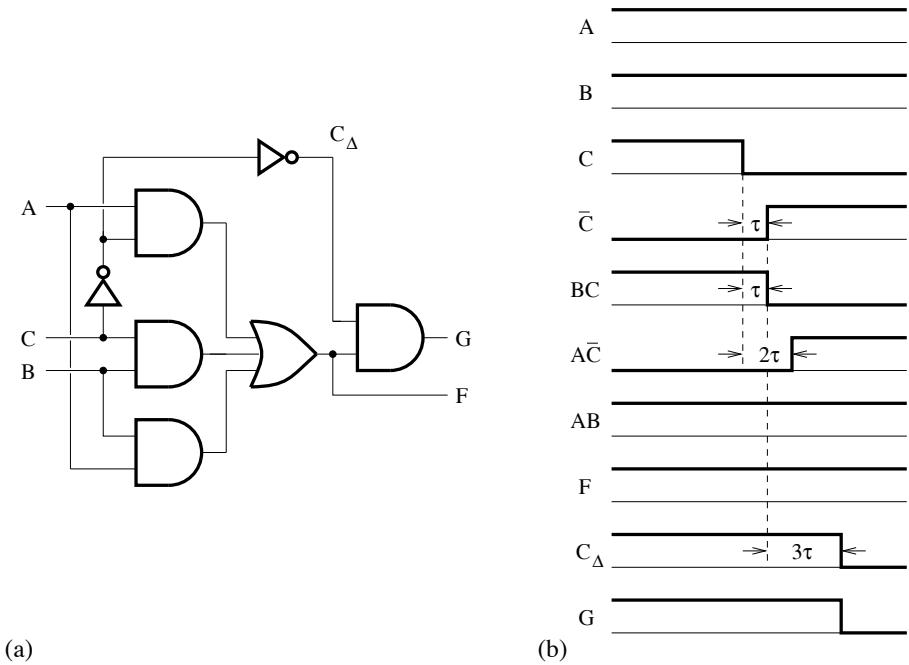


Figure 2.61. *a) Logic circuit operating without hazard and b) timing diagram*

DEMONSTRATION 2.3.– Show that $X \cdot Y + X \cdot \overline{Y} = X$. We have:

$$\begin{aligned} X \cdot Y + X \cdot \overline{Y} &= X(Y + \overline{Y}) \\ &= X \end{aligned} \qquad [2.64]$$

DEMONSTRATION 2.4.– Show that $(X + Y)(X + \overline{Y}) = X$. We have:

$$\begin{aligned} (X + Y)(X + \overline{Y}) &= X \cdot X + X \cdot \overline{Y} + X \cdot Y + Y \cdot \overline{Y} \\ &= X + X \cdot \overline{Y} + X \cdot Y \\ &= X(1 + \overline{Y} + Y) \\ &= X \end{aligned} \qquad [2.65]$$

DEMONSTRATION 2.5.– Show that $X + \overline{X} \cdot Y = X + Y$. We have:

$$\begin{aligned} X + \overline{X} \cdot Y &= (X + X \cdot Y)1 + \overline{X} \cdot Y \\ &= X + X \cdot Y + \overline{X} \cdot Y \\ &= X + Y(X + \overline{X}) \\ &= X + Y(1) \\ &= X + Y \end{aligned} \quad [2.66]$$

DEMONSTRATION 2.6.– Show that $X(\overline{X} + Y) = X \cdot Y$. We have:

$$\begin{aligned} X(\overline{X} + Y) &= X \cdot \overline{X} + X \cdot Y \\ &= 0 + X \cdot Y \\ &= X \cdot Y \end{aligned} \quad [2.67]$$

DEMONSTRATION 2.7.– Show that $X \cdot Y + \overline{X} \cdot Z = (X + Z)(\overline{X} + Y)$. We have:

$$\begin{aligned} X \cdot Y + \overline{X} \cdot Z &= X \cdot Y(1 + Z) + \overline{X} \cdot Z(1 + Y) + X \cdot \overline{X} \\ &= X \cdot Y + Y \cdot Z(X + \overline{X}) + \overline{X} \cdot Z + X \cdot \overline{X} \\ &= (X + Z)Y + (X + Z)\overline{X} \\ &= (X + Z)(\overline{X} + Y) \end{aligned} \quad [2.68]$$

DEMONSTRATION 2.8.– Show that $(X + Y)(\overline{X} + Z) = X \cdot Z + \overline{X} \cdot Y$. We have:

$$\begin{aligned} (X + Y)(\overline{X} + Z) &= X \cdot \overline{X} + X \cdot Z + \overline{X} \cdot Y + Y \cdot Z \\ &= X \cdot Z + \overline{X} \cdot Y + Y \cdot Z \\ &= X \cdot Z + \overline{X} \cdot Y + (X + \overline{X})Y \cdot Z \\ &= X \cdot Z(1 + Y) + \overline{X} \cdot Y(1 + Z) \\ &= X \cdot Z + \overline{X} \cdot Y \end{aligned} \quad [2.69]$$

DEMONSTRATION 2.9.– Show that $X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z$. We have:

$$\begin{aligned} X \cdot Y + \overline{X} \cdot Z + Y \cdot Z &= X \cdot Y + \overline{X} \cdot Z + Y \cdot Z(X + \overline{X}) \\ &= X \cdot Y + \overline{X} \cdot Z + X \cdot Y \cdot Z + \overline{X} \cdot Y \cdot Z \\ &= X \cdot Y(1 + Z) + \overline{X} \cdot Z(1 + Y) \\ &= X \cdot Y(1) + \overline{X} \cdot Z(1) \\ &= X \cdot Y + \overline{X} \cdot Z \end{aligned} \quad [2.70]$$

DEMONSTRATION 2.10.– Show that $(X+Y)(\overline{X}+Z)(Y+Z) = (X+Y)(\overline{X}+Z)$. We have:

$$\begin{aligned}(X+Y)(\overline{X}+Z)(Y+Z) &= (X\cdot Z+\overline{X}\cdot Y)(Y+Z) \qquad \text{according to [8]}\\ &= X\cdot Y\cdot Z+\overline{X}\cdot Y+X\cdot Z+\overline{X}\cdot Y\cdot Z\\ &= X\cdot Z(1+Y)+\overline{X}\cdot Y(1+Z)\\ &= X\cdot Z+\overline{X}\cdot Y\\ &= (X+Y)(\overline{X}+Z) \qquad \text{according to [7]}\end{aligned} \qquad [2.71]$$

DEMONSTRATION 2.11.– Show that $(X\cdot Y)\oplus(X\cdot Z) = X(Y\oplus Z)$. We have:

$$\begin{aligned}(X\cdot Y)\oplus(X\cdot Z) &= (X\cdot Y)(\overline{X\cdot Z})+(\overline{X\cdot Y})(X\cdot Z)\\ &= (X\cdot Y)(\overline{X}+\overline{Z})+(\overline{X}+\overline{Y})(X\cdot Z)\\ &= X\cdot Y\cdot\overline{Z}+X\cdot\overline{Y}\cdot Z\\ &= X(Y\cdot\overline{Z}+\overline{Y}\cdot Z)\\ &= X(Y\oplus Z)\end{aligned} \qquad [2.72]$$

DEMONSTRATION 2.12.– Show that $X(\overline{X}\oplus Y) = X\cdot Y$. We have:

$$\begin{aligned}X(\overline{X}\oplus Y) &= X(\overline{X}\cdot\overline{Y}+X\cdot Y)\\ &= X\cdot Y\end{aligned} \qquad [2.73]$$

DEMONSTRATION 2.13.– Show that $(X\cdot Y)\oplus(\overline{X}\cdot Z)+Y\cdot Z = (X\cdot Y)\oplus(\overline{X}\cdot Z)$. We have:

$$\begin{aligned}(X\cdot Y)\oplus(\overline{X}\cdot Z)+Y\cdot Z &= (X\cdot Y)(\overline{\overline{X}\cdot Z})+(\overline{X\cdot Y})(\overline{X}\cdot Z)+Y\cdot Z\\ &= X\cdot Y(X+\overline{Z})+(\overline{X}+\overline{Y})\overline{X}\cdot Z+Y\cdot Z\\ &= X\cdot Y\cdot\overline{Z}+\overline{X}\cdot Z(1+Z)+Y\cdot Z\\ &= X\cdot Y\cdot\overline{Z}+\overline{X}\cdot Z+(X+\overline{X})Y\cdot Z\\ &= X\cdot Y(Z+\overline{Z})+\overline{X}\cdot Z(1+Y)\\ &= X\cdot Y+\overline{X}\cdot Z\\ &= X\cdot Y(1+\overline{Z})+\overline{X}\cdot Z(1+\overline{Y})\\ &= X\cdot Y+X\cdot Y\cdot\overline{Z}+\overline{X}\cdot Z+\overline{X}\cdot\overline{Y}\cdot Z\end{aligned}$$

$$
\begin{aligned}
&= X \cdot X \cdot Y + X \cdot Y \cdot \overline{Z} + \overline{X} \cdot \overline{X} \cdot Z + \overline{X} \cdot \overline{Y} \cdot Z \\
&= X \cdot Y(X + \overline{Z}) + (\overline{X} + \overline{Y})\overline{X} \cdot Z \\
&= X \cdot Y(\overline{\overline{X} \cdot Z}) + (\overline{X \cdot Y})\overline{X} \cdot Z \\
&= (X \cdot Y) \oplus (\overline{X} \cdot Z) \qquad [2.74]
\end{aligned}
$$

2.10. Exercises

EXERCISE 2.1.– Function of n variables.

How many different logic functions can we implement using n input variables?

EXERCISE 2.2.– Analysis of a logic circuit.

Construct the truth table for the circuit shown in Figure 2.62. Using a minimum number of NAND gates, propose a circuit to implement F.

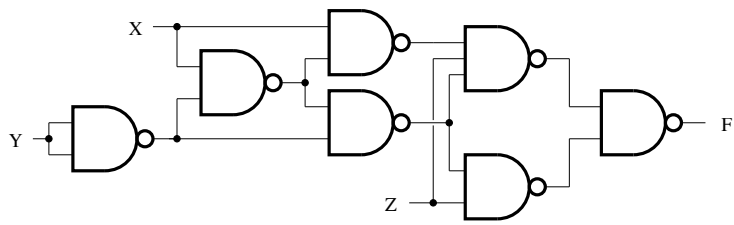


Figure 2.62. *Logic circuit*

EXERCISE 2.3.– Simplification of logic functions.

Simplify the following logic expressions:

a) $X + \overline{Y} + \overline{X} \cdot Y + (X + \overline{Y}) \cdot \overline{X} \cdot Y$

b) $X \cdot Y \cdot Z + X \cdot \overline{Y} \cdot Z + \overline{X} \cdot Y \cdot Z + \overline{X} \cdot Y \cdot \overline{Z}$

c) $\overline{\overline{\overline{X} \cdot Y} \cdot \overline{X \cdot Z}}$

d) $(X + \overline{Y})(\overline{X} + Z)(Y + \overline{Z})$

e) $(W + X + Y \cdot Z)(\overline{W} + X)(\overline{X} + Y)$

f) $W \cdot X \cdot \overline{Z} + \overline{X} \cdot Y \cdot Z + W \cdot X \cdot \overline{Y} + X \cdot Y \cdot Z + \overline{W} \cdot Y \cdot Z$

g) $\overline{W} \cdot X \cdot Z + W \cdot Z + X \cdot Y \cdot \overline{Z} + \overline{W} \cdot X \cdot Y$

h) $(X + Y + Z)(\overline{X} + Y + Z)(\overline{X} + Y + \overline{Z})$

EXERCISE 2.4.– Equivalent logic functions.

Verify the following equalities:

a) $\overline{A} \cdot B \cdot \overline{C} + A \cdot C + \overline{B} \cdot C = C \oplus (\overline{A} \cdot B)$

b) $A \oplus B \oplus (A + B) + \overline{A} \cdot \overline{B} = A \odot B$

c) $(A + B) \odot (\overline{A} + C) + A = A + B$

d) $(A \odot B) \oplus (A \cdot \overline{B}) + A \cdot \overline{B} = A + \overline{B}$

e) $\overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot D + A \cdot B \cdot \overline{C} \cdot D + A \cdot C \cdot \overline{D} + A \cdot \overline{B} \cdot C = A \cdot C \oplus \overline{A} \cdot \overline{C} \oplus B \cdot D$

f) $A \cdot B \cdot \overline{D} + \overline{B} \cdot C \cdot D + A \cdot B \cdot \overline{C} + B \cdot C \cdot D + \overline{A} \cdot C \cdot D = A \cdot B + C \cdot D$

EXERCISE 2.5.– Simplification of the functions using the Karnaugh map method.

Using the Karnaugh map method, simplify each of the following logic functions:

$$E(A, B, C, D) = \sum mm(0, 2, 3, 4, 5, 8, 11, 12, 13, 14, 15) \quad [2.75]$$

$$F(A, B, C, D) = \sum m(0, 1, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15) \quad [2.76]$$

$$G(A, B, C, D) = \sum m(0, 2, 4, 5, 10, 12, 15) + \sum x(8, 14) \quad [2.77]$$

$$H(A, B, C, D) = \sum mm(1, 3, 6, 8, 11, 14) + \sum x(2, 5, 12, 13, 15) \quad [2.78]$$

EXERCISE 2.6.– Analysis of the circuit with NAND/NOR gates.

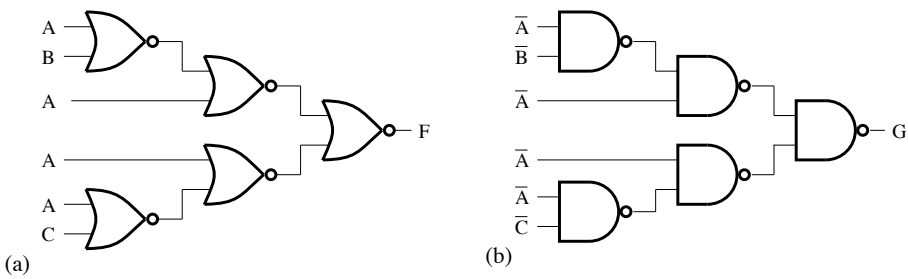


Figure 2.63. *a) Circuit with NAND gates and b) circuit with NOR gates*

Determine the logic function F implemented by the circuit shown in Figure 2.63(a). Determine the logic function G implemented by the circuit shown in Figure 2.63(b). Verify that $F = \overline{G}$.

EXERCISE 2.7.– Canonical forms of a logic function.

Let us consider the logic functions represented in Figure 2.64.

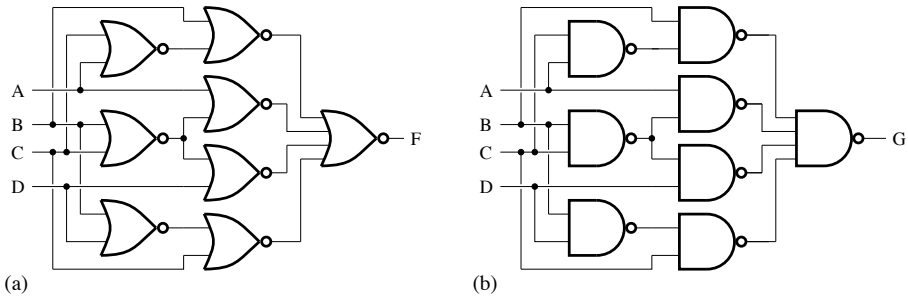


Figure 2.64. *a) Circuit with NOR gates and b) circuit with NAND gates*

Determine the logic expressions for the outputs F and G.

Express the logic function F as a sum of products.

Give the product-of-sums form of G.

EXERCISE 2.8.– Simplification of logic circuits.

Simplify each of the logic circuits shown in Figure 2.65.

EXERCISE 2.9.– Implementation of the function H.

Let us consider the three-variable logic function defined as follows:

$$H(A,B,C) = A \cdot \overline{B} + \overline{A} \cdot B + B \cdot \overline{C} \qquad [2.79]$$

Assuming the input variables are A, B and C, propose a logic circuit using only 2-input NAND gates to implement the function H.

EXERCISE 2.10.– Implementation of the logic function Y.

Using the minimum number of logic gates (NOT, AND, OR) to realize the following logic function:

$$Y = (A \cdot B) \oplus (\overline{B} \cdot \overline{C}) \qquad [2.80]$$

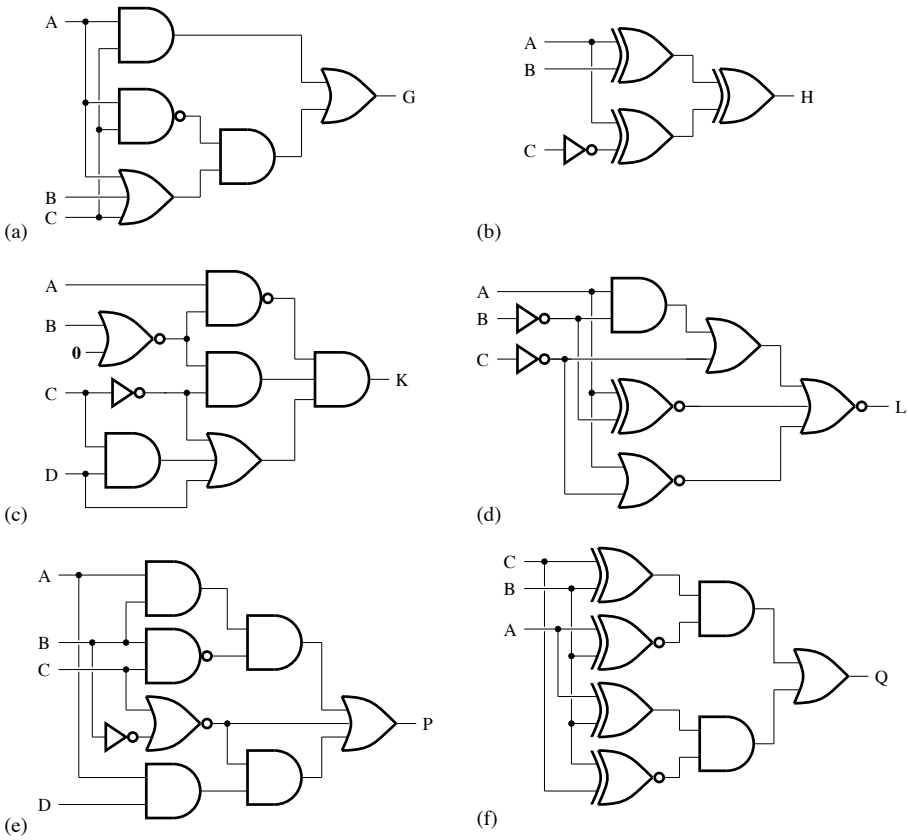


Figure 2.65. *Logic circuits*

EXERCISE 2.11.– Implementation of circuits with NAND/NOR gates.

Let us consider the following logic functions:

$$P(A,B,C,D) = A \cdot B + \overline{A} \cdot \overline{C} \cdot \overline{D} \quad [2.81]$$

$$\text{and} Q(A,B,C,D) = (\overline{A} + \overline{B} + \overline{C})(A + D) \quad [2.82]$$

Assuming that the input variables A, B, C, and D, as well as their complements, are available, propose a logic circuit based on 2-input NAND gates and a logic circuit based on 2-input NOR gates for each of these functions.

EXERCISE 2.12.– Control circuit for a switcher.

We wish to switch in four directions packages identified by an eight-bit binary code $I_7\, I_6\, I_5\, I_4\, I_3\, I_2\, I_1\, I_0$, where I_7 is the most significant bit. The signal D_0 is set at 1 if no direction is chosen and the switching, based on the code of each packet, is carried out as follows:

$$D_1 = 1 \text{ if } 32 \leq \mathbf{N} \leq 63,$$
$$D_2 = 1 \text{ if } 64 \leq \mathbf{N} \leq 127,$$
$$D_3 = 1 \text{ if } 128 \leq \mathbf{N} \leq 159,$$
$$D_4 = 1 \text{ if } 192 \leq \mathbf{N} \leq 255,$$

The number $\mathbf{N}$ being the decimal number corresponding to the code $I_7\, I_6\, I_5\, I_4\, I_3\, I_2\, I_1\, I_0$.

a) – Determine the Boolean expression for the logic function of selection (D_1, D_2, D_3 and D_4) for each direction.

– Deduce the Boolean expression for the function D_0.

b) Implement these functions using only inverters and NAND gates with at most three inputs.

2.11. Solutions

SOLUTION 2.1.– Function with n variables.

With n variables, the truth table has 2^n columns and we can choose any number of 2^n bits for each column. There are, thus, 2^{2^n} different functions with n variables.

n	1	2	3	4
Number of functions	4	16	256	65, 536

SOLUTION 2.2.– Analysis of a logic circuit.

By analyzing the logic circuit, the equation for the output F can be obtained as follows:

$$F = \overline{\overline{X \cdot \overline{X \cdot \overline{Y}} \cdot \overline{X \cdot \overline{Y}} \cdot \overline{Y} \cdot Z} \cdot \overline{\overline{X \cdot \overline{Y}} \cdot \overline{Y} \cdot Z}}$$
$$= (Y + \overline{Y} \cdot X)Z(1 + \overline{X} + X \cdot \overline{Y}) \;=\; (X + Y)Z = X \cdot Z + Y \cdot Z$$

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 2.14. *Truth table*

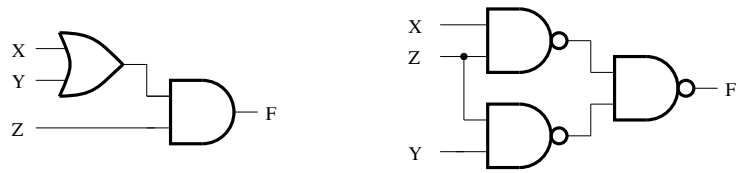


Figure 2.66. *Simplified circuits*

Table 2.14 shows the truth table constructed based on the logic equation for the output F.

The circuits obtained upon simplification are represented in Figure 2.66.

SOLUTION 2.3.– Simplification of logic expressions.

The simplifications are carried out by using Boolean algebra theorems.

a) $X + \overline{Y} + \overline{X} \cdot Y + (X + \overline{Y}) \cdot \overline{X} \cdot Y = X + \overline{Y} + Y = 1$

b) $X \cdot Y \cdot Z + X \cdot \overline{Y} \cdot Z + \overline{X} \cdot Y \cdot Z + \overline{X} \cdot Y \cdot \overline{Z} = \overline{X} \cdot Y + X \cdot Z$

c) $\overline{\overline{\overline{X} \cdot Y} \cdot \overline{X \cdot Z}} = \overline{X} \cdot Y + X \cdot Z$

d) $(X + \overline{Y})(\overline{X} + Z)(Y + \overline{Z}) = X \cdot Y \cdot Z + \overline{X} \cdot \overline{Y} \cdot \overline{Z}$

e) $(W + X + Y \cdot Z)(\overline{W} + X)(\overline{X} + Y) = X \cdot Y + \overline{W} \cdot Y \cdot Z$

f)

$$\begin{aligned} W \cdot X \cdot \overline{Z} + \overline{X} \cdot Y \cdot Z + W \cdot X \cdot \overline{Y} + X \cdot Y \cdot Z & \\ + \overline{W} \cdot Y \cdot Z &= W \cdot X(\overline{Y} + \overline{Z}) + Y \cdot Z \\ &= W \cdot X(\overline{Y} + \overline{Z}) + \overline{\overline{Y} + \overline{Z}} \\ &= W \cdot X + Y \cdot Z \end{aligned}$$

g) $\overline{W} \cdot X \cdot Z + W \cdot Z + X \cdot Y \cdot \overline{Z} + \overline{W} \cdot X \cdot Y = X \cdot Y + X \cdot Z + W \cdot Z$

h)

$$\begin{aligned} (X + Y + Z)(\overline{X} + Y + Z)(\overline{X} + Y + \overline{Z}) &= (Y + Z)(\overline{X} + Y + \overline{Z}) \\ &\quad \text{because} \quad (A + B)(A + \overline{B}) = A \\ &= Z(\overline{X} + Y) + Y \cdot \overline{Z} \\ &\quad \text{because} \quad (A + B)(\overline{A} + C) = A \cdot C + \overline{A} \cdot C \\ &= \overline{X} \cdot Z + Y \end{aligned}$$

SOLUTION 2.4.– Equivalent logic functions.

Verification of the following equalities:

a) $\overline{A} \cdot B \cdot \overline{C} + A \cdot C + \overline{B} \cdot C = \overline{A} \cdot B \cdot \overline{C} + (\overline{\overline{A} \cdot B})C$

$$= C \oplus (\overline{A} \cdot B)$$

b) $A \oplus B \oplus (A + B) + \overline{A} \cdot \overline{B} = A \oplus [\overline{B} \oplus (\overline{A} \cdot \overline{B})] + \overline{A} \cdot \overline{B}$

$$\begin{aligned} &= A \oplus [\overline{B}(1 \oplus \overline{A})] + \overline{A} \cdot \overline{B} \\ &= A \oplus (A \cdot \overline{B}) + \overline{A} \cdot \overline{B} \\ &= A \cdot B + \overline{A} \cdot \overline{B} \\ &= A \odot B \end{aligned}$$

c) $(A+B)\odot(\overline{A}+C)+A=(A+B)(\overline{A}+C)+(\overline{A}\cdot\overline{B})(A\cdot\overline{C})+A$

$$
\begin{aligned}
&= \overline{A}\cdot B + A\cdot C + B\cdot C + A \\
&= A(1+C)+B(1+C) \\
&\qquad \text{because} \quad A+\overline{A}\cdot B = A+B \\
&= A+B
\end{aligned}
$$

d) $(A\odot B)\oplus(A\cdot\overline{B})+A\cdot B=(A\odot B)(\overline{A\cdot\overline{B}})+(A\oplus B)(A\cdot\overline{B})+A\cdot B$

$$
\begin{aligned}
&= (A\odot B)(\overline{A\cdot\overline{B}})+A\cdot\overline{B}+A\cdot B \\
&= A\odot B + A\cdot\overline{B}+A\cdot B \\
&\quad \text{because} \quad (A\odot B)(\overline{A\cdot\overline{B}})+A\cdot\overline{B} = A\odot B + A\cdot\overline{B} \\
&= A\cdot B+\overline{A}\cdot\overline{B}+A(B+\overline{B}) \\
&= A\cdot B+\overline{A}\cdot\overline{B}+A \\
&= A\cdot B+A+\overline{B} \\
&= A+\overline{B}
\end{aligned}
$$

e) $\overline{A}\cdot\overline{B}\cdot\overline{C}+\overline{A}\cdot\overline{C}\cdot\overline{D}+\overline{A}\cdot B\cdot C\cdot D+A\cdot B\cdot\overline{C}\cdot D+A\cdot C\cdot\overline{D}$
$+A\cdot\overline{B}\cdot C$

$$
\begin{aligned}
&= \overline{A}\cdot\overline{C}(\overline{B}+\overline{D})+(\overline{A}\cdot C+A\cdot\overline{C})B\cdot D+A\cdot C(\overline{B}+\overline{D}) \\
&= (\overline{A}\cdot C+A\cdot\overline{C})B\cdot D+(A\cdot C+\overline{A}\cdot\overline{C})(\overline{B}+\overline{D}) \\
&= (\overline{A}\cdot C\oplus A\cdot\overline{C})B\cdot D+(A\cdot C\oplus\overline{A}\cdot\overline{C})(\overline{B\cdot D}) \\
&\qquad \text{because} \quad (\overline{A}\cdot C)(A\cdot\overline{C})=0 \quad \text{and} \quad (A\cdot C)(\overline{A}\cdot\overline{C})=0 \\
&= (\overline{A\cdot C\oplus\overline{A}\cdot\overline{C}})B\cdot D+(A\cdot C\oplus\overline{A}\cdot\overline{C})(\overline{B\cdot D}) \\
&= A\cdot C\oplus\overline{A}\cdot\overline{C}\oplus B\cdot D
\end{aligned}
$$

f) $A \cdot B \cdot \overline{D} + \overline{B} \cdot C \cdot D + A \cdot B \cdot \overline{C} + B \cdot C \cdot D + \overline{A} \cdot C \cdot D$

$$= A \cdot B \cdot \overline{D} + C \cdot D(\overline{B} + B + \overline{A}) + A \cdot B \cdot \overline{C}$$

$$= A \cdot B \cdot \overline{D} + C \cdot D + A \cdot B \cdot \overline{C}$$

$$= A \cdot B(\overline{C} + \overline{D}) + C \cdot D$$

$$= A \cdot B(\overline{C} + \overline{D}) + \overline{\overline{C} + \overline{D}}$$

$$= A \cdot B + C \cdot D$$

because $\quad X + \overline{X} \cdot Y = X + Y$

SOLUTION 2.5.– Simplification of functions using the Karnaugh map method.

– Function E:

The function $E(A, B, C, D)$ has three minimized forms that can be obtained from the Karnaugh maps, as shown in Figure 2.67, as follows:

$$E(A, B, C, D) = A \cdot B + B \cdot \overline{C} + \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C + A \cdot C \cdot D \quad (a)$$

$$= A \cdot B + B \cdot \overline{C} + \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C + \overline{B} \cdot C \cdot D \quad (b)$$

$$= A \cdot B + B \cdot \overline{C} + \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{D} + \overline{B} \cdot C \cdot D \quad (c)$$

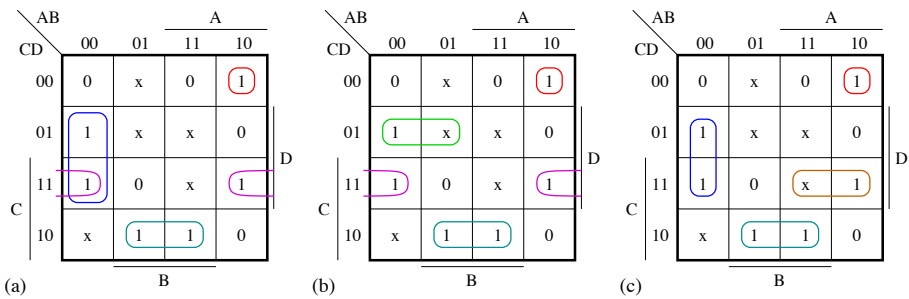


Figure 2.67. *Karnaugh maps (function E)*

– Function F:

In the case of the function $F(A, B, C, D)$, there are six possible solutions. Figure 2.68 shows the Karnaugh maps corresponding to the different solutions. Thus:

$$
\begin{aligned}
F(A,B,C,D) &= \overline{C}\cdot\overline{D} + \overline{B}\cdot D + B\cdot C + A\cdot B \quad (a)\\
&= \overline{C}\cdot\overline{D} + \overline{B}\cdot D + B\cdot C + A\cdot \overline{C} \quad (b)\\
&= \overline{C}\cdot\overline{D} + \overline{B}\cdot D + B\cdot C + A\cdot D \quad (c)\\
&= \overline{B}\cdot\overline{C} + B\cdot\overline{D} + C\cdot D + A\cdot B \quad (d)\\
&= \overline{B}\cdot\overline{C} + B\cdot\overline{D} + C\cdot D + A\cdot \overline{C} \quad (e)\\
&= \overline{B}\cdot\overline{C} + B\cdot\overline{D} + C\cdot D + A\cdot D \quad (f)
\end{aligned}
$$

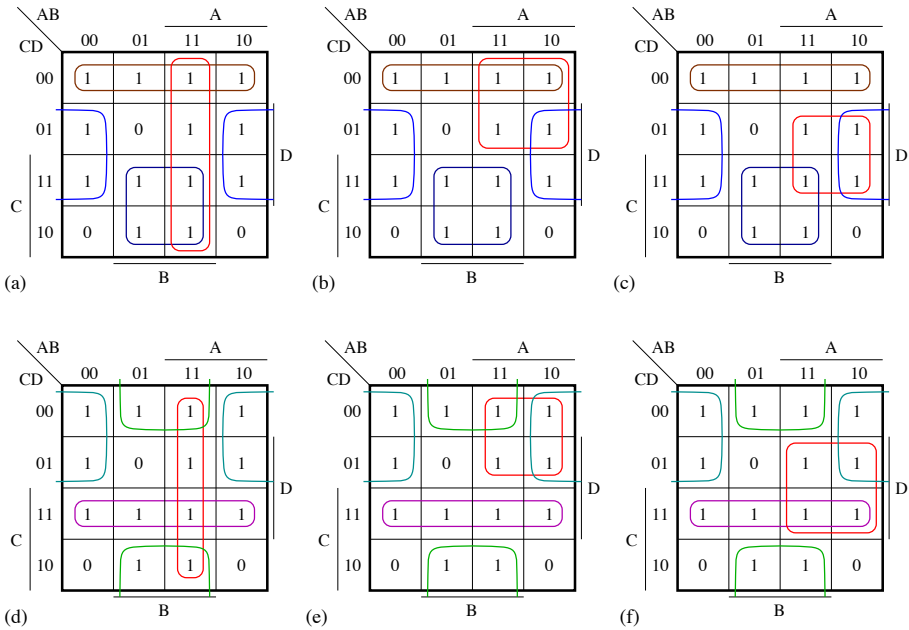


Figure 2.68. *Karnaugh maps (function F)*

– Function G:

The function G has two minimum forms corresponding to the Karnaugh maps, as shown in Figure 2.69. We can, thus, write:

$$G(A,B,C,D) = \overline{A}\cdot B\cdot \overline{C} + A\cdot B\cdot C + \overline{B}\cdot \overline{D} + \overline{C}\cdot \overline{D} \quad (a)$$
$$= \overline{A}\cdot B\cdot \overline{C} + A\cdot B\cdot C + \overline{B}\cdot \overline{D} + A\cdot \overline{D} \quad (b)$$

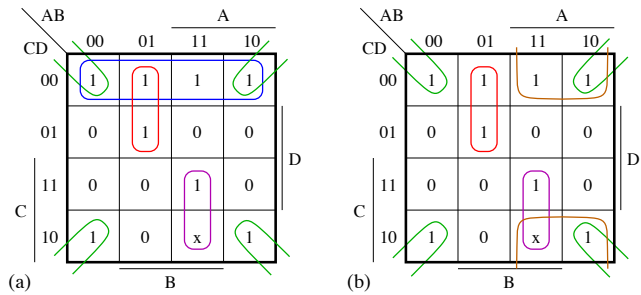


Figure 2.69. *Karnaugh maps (function G)*

– Function H:

As each Karnaugh map in Figure 2.70 corresponds to a minimal form, we obtain the following three logic expressions:

$$H(A,B,C,D) = A\cdot \overline{C}\cdot \overline{D} + B\cdot C\cdot \overline{D} + \overline{A}\cdot \overline{B}\cdot D + \overline{B}\cdot C\cdot D \quad (a)$$
$$= A\cdot \overline{C}\cdot \overline{D} + B\cdot C\cdot \overline{D} + \overline{A}\cdot \overline{C}\cdot D + \overline{B}\cdot C\cdot D \quad (b)$$
$$= A\cdot \overline{C}\cdot \overline{D} + B\cdot C\cdot \overline{D} + \overline{A}\cdot \overline{B}\cdot D + A\cdot C\cdot D \quad (c)$$

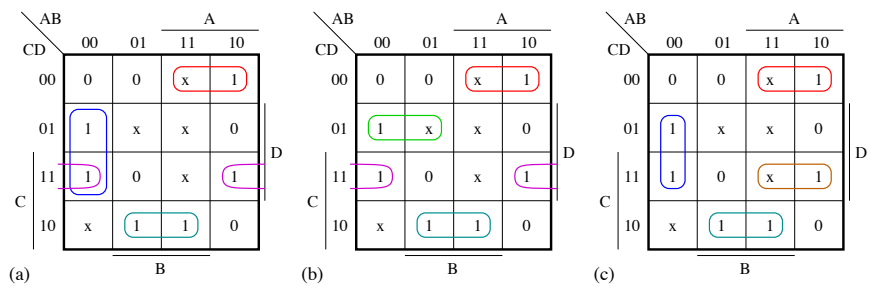


Figure 2.70. *Karnaugh maps (function H)*

SOLUTION 2.6.– Analysis of circuits with NAND/NOR gates.

For the circuit shown in Figure 2.63(a), we have:

$$F = \overline{\overline{A + \overline{A + B}} + \overline{A + \overline{A + C}}} \quad [2.83]$$

$$= (A + \overline{A + B})(A + \overline{A + C})$$

$$= (A + \overline{A} \cdot \overline{B})(A + \overline{A} \cdot \overline{C})$$

$$= (A + \overline{B})(A + \overline{C}) \quad [2.84]$$

$$= A(1 + \overline{B} + \overline{C}) + \overline{B} \cdot \overline{C}$$

$$= A + \overline{B} \cdot \overline{C} \quad [2.85]$$

For the circuit represented in Figure 2.63(b), we have:

$$G = \overline{\overline{A} \cdot \overline{\overline{A} \cdot \overline{B}} \cdot \overline{A} \cdot \overline{\overline{A} \cdot \overline{C}}} \quad [2.86]$$

$$= \overline{A} \cdot \overline{\overline{A} \cdot \overline{B}} + \overline{A} \cdot \overline{\overline{A} \cdot \overline{C}}$$

$$= \overline{A}(A + B) + \overline{A}(A + C)$$

$$= \overline{A}(B + C) \quad [2.87]$$

It can then be verified that:

$$\overline{G} = \overline{\overline{A}(B + C)}$$

$$= A + \overline{B} \cdot \overline{C}$$

$$= F \quad [2.88]$$

SOLUTION 2.7.– Canonical forms of a logic function.

By analyzing each logic circuit, we can write:

$$F = \overline{\overline{A + \overline{C} + B} + \overline{A + \overline{B + \overline{C}}} + \overline{\overline{B + \overline{C}} + D} + \overline{C + \overline{B + D}}} \quad [2.89]$$

$$= (\overline{A + \overline{C} + B})(A + \overline{B + \overline{C}})(\overline{B + \overline{C}} + D)(C + \overline{B + D})$$

$$= (\overline{A} \cdot \overline{C} + B)(A + \overline{B} \cdot \overline{C})(\overline{B} \cdot \overline{C} + D)(C + \overline{B} \cdot \overline{D})$$

$$= (A \cdot B + \overline{A} \cdot \overline{B} \cdot \overline{C})(C \cdot D + \overline{B} \cdot \overline{C} \cdot \overline{D})$$

$$= \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot C \cdot D \quad [2.90]$$

$$= \sum m(0, 15) \quad [2.91]$$

and

$$G = \overline{\overline{\overline{A \cdot \overline{C}} \cdot B} \cdot \overline{A \cdot \overline{B \cdot \overline{C}}} \cdot \overline{\overline{B \cdot \overline{C}} \cdot D} \cdot \overline{C \cdot \overline{B \cdot D}}} \quad [2.92]$$

$$= \overline{(A \cdot C + \overline{B})(\overline{A} + B \cdot C)(B \cdot C + \overline{D})(\overline{C} + B \cdot D)}$$

$$= \overline{(\overline{A} \cdot \overline{B} + A \cdot B \cdot C)(\overline{C} \cdot \overline{D} + B \cdot C \cdot D)}$$

$$= \overline{\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot C \cdot D}$$

$$= (A + B + C + D)(\overline{A} + \overline{B} + \overline{C} + \overline{D}) \quad [2.93]$$

$$= \prod m(0, 15) \quad [2.94]$$

It must be noted that $G = \overline{F}$.

SOLUTION 2.8.– Simplification of logic circuits.

Figure 2.71 depicts circuits obtained upon simplification.

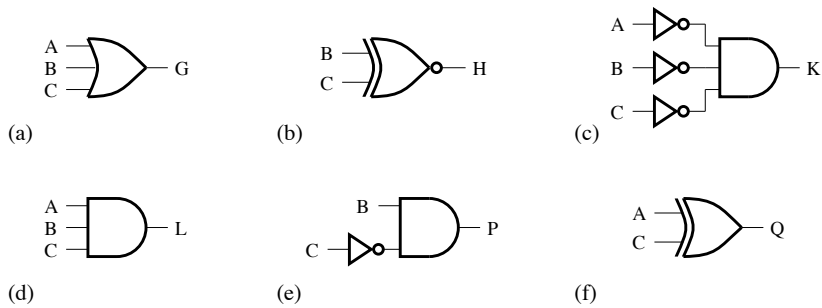


Figure 2.71. *Logic circuits*

– circuit (a):

$$G = A \cdot C + \overline{A \cdot C}(A + B + C) \quad [2.95]$$

$$= A \cdot C + A + B + C$$

$$= A(C + 1) + B + C$$

$$= A + B + C \quad [2.96]$$

– circuit (b):

$$H = (A \oplus B) \oplus (A \oplus \overline{C}) \quad [2.97]$$
$$= (A \oplus A) \oplus (B \oplus \overline{C})$$
$$= 0 \oplus (B \oplus \overline{C})$$
$$= B \oplus \overline{C} = \overline{B \oplus C} \quad [2.98]$$

– circuit (c):

$$K = \overline{A \cdot \overline{B}} \cdot \overline{B} \cdot \overline{C}(\overline{C} + D) \quad [2.99]$$
$$= (\overline{A} + B)\overline{B} \cdot \overline{C}(\overline{C} + D)$$
$$= \overline{A} \cdot \overline{B} \cdot \overline{C}(\overline{C} + D)$$
$$= \overline{A} \cdot \overline{B} \cdot \overline{C}(1 + D)$$
$$= \overline{A} \cdot \overline{B} \cdot \overline{C} \quad [2.100]$$

– circuit (d):

$$L = \overline{A \cdot \overline{B} + \overline{C} + \overline{A \oplus \overline{B}} + \overline{A + \overline{C}}} \quad [2.101]$$
$$= \overline{A \cdot \overline{B} + \overline{C} + \overline{A} \cdot C + A \oplus B}$$
$$= \overline{A \cdot \overline{B} + \overline{A} + \overline{C} + A \oplus B}$$
$$= \overline{\overline{A} + \overline{B} + \overline{C} + A \oplus B}$$
$$= \overline{\overline{A}(1 + B) + \overline{B}(1 + A) + \overline{C}}$$
$$= A \cdot B \cdot C \quad [2.102]$$

– circuit (e):

$$P = A \cdot B \cdot \overline{B \cdot C} + A \cdot D \cdot \overline{\overline{B} + C} + \overline{\overline{B} + C} \quad [2.103]$$
$$= A \cdot B(\overline{B} + \overline{C}) + A \cdot D \cdot B \cdot \overline{C} + B \cdot \overline{C}$$
$$= (A + A \cdot D + 1)B \cdot \overline{C}$$
$$= B \cdot \overline{C} \quad [2.104]$$

– circuit (f):

$$Q = (A \oplus B)\overline{B \oplus C} + \overline{A \oplus B}(B \oplus C) \quad [2.105]$$

$$= (A \oplus B) \oplus (B \oplus C)$$

$$= (A \oplus C) \oplus (B \oplus B)$$

$$= (A \oplus C) \oplus 0$$

$$= A \oplus C \quad [2.106]$$

SOLUTION 2.9.– Implementation of the function H.

The function H can be written as follows:

$$H = A \cdot \overline{B} + \overline{A} \cdot B + B \cdot \overline{C} \quad [2.107]$$

$$= A \cdot \overline{B} + (\overline{A} + \overline{C})B$$

$$= A(\overline{A} + \overline{B}) + (\overline{A} + \overline{C})B$$

$$= A \cdot \overline{A \cdot B} + \overline{A \cdot C} \cdot B$$

$$= \overline{\overline{A \cdot \overline{A \cdot B}} \cdot \overline{\overline{A \cdot C} \cdot B}} \quad [2.108]$$

Using equation [2.108], the logic circuit based on NAND gates shown in Figure 2.72 can be obtained.

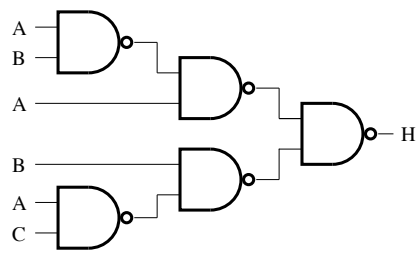


Figure 2.72. *Implementation of H: logic circuit based on NAND gates*

SOLUTION 2.10.– Implementation of the logic function Y.

Observing that:

$$Y = (A \cdot B) \oplus (\overline{B} \cdot \overline{C}) = A \cdot B + \overline{B} \cdot \overline{C} = A \cdot B + \overline{B + C} \quad [2.109]$$

because $A \cdot B \cdot \overline{B} \cdot \overline{C} = 0$, the function Y can be implemented as shown in Figure 2.73.

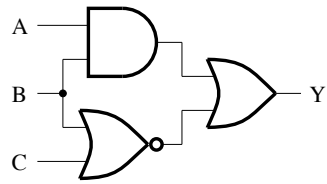


Figure 2.73. *Implementation of the function Y*

SOLUTION 2.11.– Implementation of NAND/NOR gates.

Figures 2.74 and 2.75 depict the circuits corresponding to the logic functions P and Q, respectively.

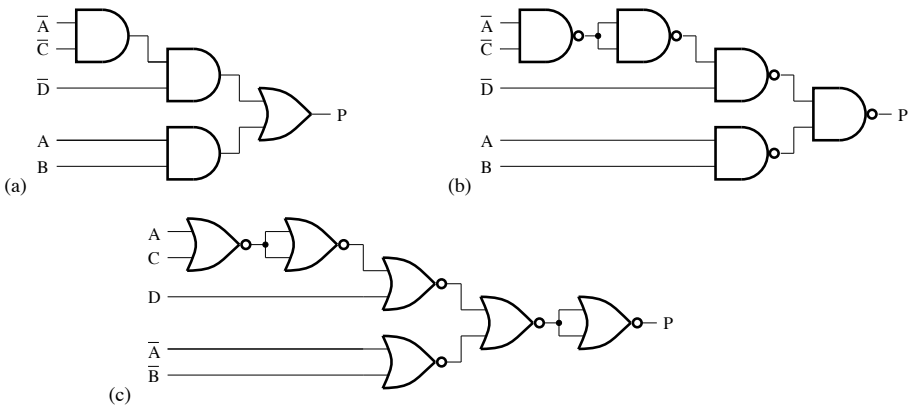


Figure 2.74. *Implementation of P: a) logic circuit using AND and OR gates; b) logic circuit based on NAND gates; c) logic circuit based on NOR gates*

SOLUTION 2.12.– Control circuit for a switcher.

The following decimal-to-binary conversion makes it possible to determine the bits I_7, I_6, I_5, I_4, I_3, I_2, I_1 and I_0:

$$\begin{aligned} 32_{10} &= 00100000_2 & 128_{10} &= 10000000_2 \\ 63_{10} &= 00111111_2 & 159_{10} &= 10011111_2 \\ 64_{10} &= 01000000_2 & 192_{10} &= 11000000_2 \\ 127_{10} &= 01111111_2 & 255_{10} &= 11111111_2 \end{aligned}$$

The truth table for the control circuit may be constructed as shown in Table 2.15.

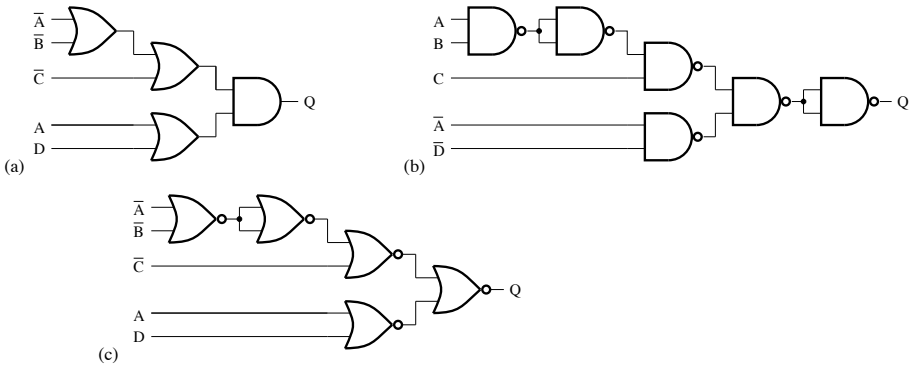


Figure 2.75. *Implementation of Q: a) logic circuit using OR and AND gates; b) logic circuit based on NAND gates; c) logic circuit based on NOR gates*

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Functions	
0	0	1	0	0	0	0	0	32	D_1
0	0	1	1	1	1	1	1	63	
0	1	0	0	0	0	0	0	64	D_2
0	1	1	1	1	1	1	1	127	
1	0	0	0	0	0	0	0	128	D_3
1	0	0	1	1	1	1	1	159	
1	1	0	0	0	0	0	0	192	D_4
1	1	1	1	1	1	1	1	255	

$D_1 = \overline{I_7}\,\overline{I_6}\,I_5$

$D_2 = \overline{I_7}\,I_6$

$D_3 = I_7\,\overline{I_6}\,\overline{I_5}$

$D_4 = I_7\,I_6$

Table 2.15. *Truth table*

For each selection range, the output function depends solely on the bits that do not change logic state. We thus have:

$$D_1 = \overline{I_7} \cdot \overline{I_6} \cdot I_5 \quad D_2 = \overline{I_7} \cdot I_6 \quad D_3 = I_7 \cdot \overline{I_6} \cdot \overline{I_5} \quad D_4 = I_7 \cdot I_6$$

and

$$D_0 = \overline{D_4 + D_3 + D_2 + D_1}$$

The output function, D_0, can also be written as follows:

$$D_0 = \overline{D_4 + D_3 + D_2 + D_1} \qquad [2.110]$$

$$= \overline{I_7 \cdot I_6 + I_7 \cdot \overline{I_6} \cdot \overline{I_5} + \overline{I_7} \cdot I_6 + \overline{I_7} \cdot \overline{I_6} \cdot I_5}$$

$$= \overline{I_7(I_6 + \overline{I_6} \cdot \overline{I_5}) + \overline{I_7}(I_6 + \overline{I_6} \cdot I_5)}$$

$$= \overline{I_7(I_6 + \overline{I_5}) + \overline{I_7}(I_6 + I_5)}$$

$$= \overline{I_7\overline{I_5} + \overline{I_7}I_5 + I_6(\overline{I_7} + I_7)}$$

$$= \overline{I_7 \oplus I_5 + I_6}$$

$$= (I_7 \odot I_5)\overline{I_6}$$

$$= I_7 \cdot \overline{I_6} \cdot I_5 + \overline{I_7} \cdot \overline{I_6} \cdot \overline{I_5} \qquad [2.111]$$

The implementation of the control circuit is represented in Figure 2.76.

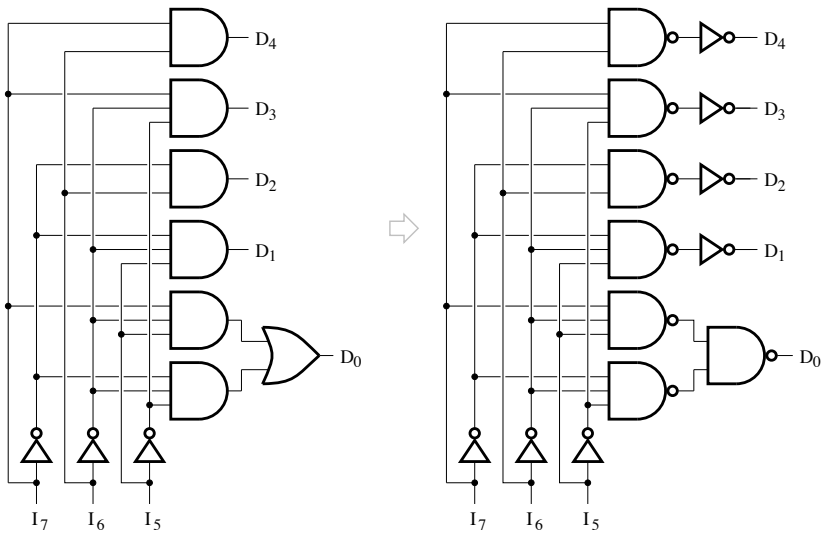


Figure 2.76. *Logic circuits*

3

Function Blocks of Combinational Logic

3.1. Introduction

Circuits are used in combinatorial logic to carry out operations such as data manipulation and selection, coding, decoding and error detection. Among the basic components listed as being necessary to implement them, there are multiplexers, demultiplexers, encoders, decoders and shifters.

3.2. Multiplexer

A multiplexer (MUX) is a logic circuit that allows for switching the data present at any one of its inputs toward its single output. Thus, it generally has 2^n data inputs, n select lines and one output.

A 2-to-1 multiplexer (or 2:1 multiplexer) can be implemented as shown in Figure 3.1(a). It can be represented by one of the symbols given in Figures 3.1(b) and 3.1(c). The logic equation of the 2:1 multiplexer is given by:

$$Y = \overline{S} \cdot D_0 + S \cdot D_1 \qquad [3.1]$$

– if $S = 0$, we have $Y = D_0$;

– if $S = 1$, we have $Y = D_1$.

A 2-to-1 multiplexer thus operates as a commutator. Its working principle is illustrated in the schematic diagram shown in Figure 3.2. By expressing the output as a function of the inputs D_0 and D_1, the size of the truth table for the 2-to-1 multiplexer can be reduced, as illustrated in Table 3.1.

Digital Electronics 1: *Combinational Logic Circuits*, First Edition. Tertulien Ndjountche.
© ISTE Ltd 2016. Published by ISTE Ltd and John Wiley & Sons, Inc.

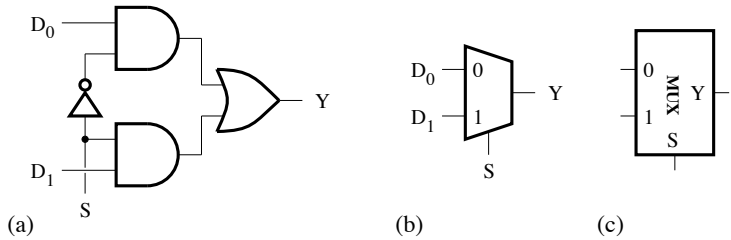


Figure 3.1. *2-to-1 multiplexer: a) logic circuit and b) and c) symbols*

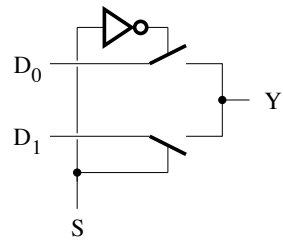


Figure 3.2. *Schematic diagram of the 2-to-1 multiplexer*

S	Y
0	D_0
1	D_1

Table 3.1. *Truth table of the 2-to-1 multiplexer*

Multiplexers offered by integrated circuit manufacturers most often have an active-low enable input. Figures 3.3(a) and 3.3(b) show, respectively, the circuit and symbol for a 2-to-1 multiplexer with an active-low enable input. The logic equation for the output is given by:

$$Y = \overline{EN}(\overline{S} \cdot D_0 + S \cdot D_1) \qquad [3.2]$$

Equation [3.2] may be translated to a truth table with entered input variables, as shown in Table 3.2.

The output of a three-state buffer reflects the input logic level or is isolated from the input depending on the logic level of the selection signal. A 2-to-1 multiplexer can thus be implemented by connecting the outputs of two three-state buffers whose selection

signals are complementary, as shown in Figure 3.4. In the ideal case, the multiplexer operation is governed by the truth table represented in Table 3.3. However, if the selection signals overlap, due to propagation delays for instance, one of the outputs Y_0 and Y_1 may be set at 0 while the other is at 1, thus forcing the output Y to assume an indeterminate state.

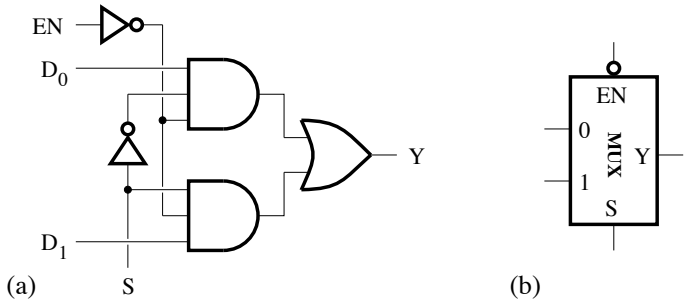


Figure 3.3. *2-to-1 multiplexer with an active-low enable input*

EN	S	Y
1	x	0
0	0	D_0
0	1	D_1

Table 3.2. *Truth table of a 2:1 multiplexer*

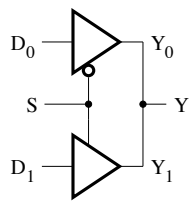


Figure 3.4. *2-to-1 multiplexer based on three-state buffers*

A 4-to-1 multiplexer can be implemented using logic gates as shown in Figure 3.5(a) or using 2-to-1 multiplexers configured as shown in Figure 3.5(b). Its

symbol is shown in Figure 3.5(c). Analyzing the circuit for the 4-to-1 multiplexer, we have:

$$Y = \overline{S_1} \cdot \overline{S_0} \cdot D_0 + \overline{S_1} \cdot S_0 \cdot D_1 + S_1 \cdot \overline{S_0} \cdot D_2 + S_1 \cdot S_0 \cdot D_3 \qquad [3.3]$$

where D_0, D_1, D_2 and D_3 represent the data input and S_0 and S_1 are the selection lines.

S	Y_0	Y_1	Y
0	D_0	z	D_0
1	z	D_1	D_1

Table 3.3. *Truth table of a 2-to-1 multiplexer*

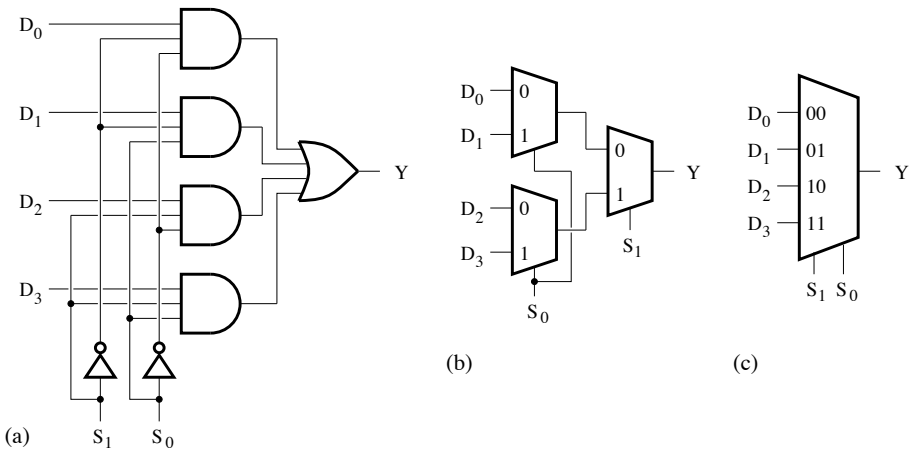


Figure 3.5. *4-to-1 multiplexers implemented using a) logic gates and b) 2-to-1 multiplexers; c) symbol*

Figure 3.6(a) depicts the logic circuit for a 4-to-1 multiplexer whose output is set to zero when $EN = 1$. Figure 3.6(b) depicts the logic circuit for a 4-to-1 multiplexer whose output is set to high impedance state when $EN = 1$. Figure 3.6(c) shows the symbol for a 4-to-1 multiplexer with enable signal. The output variable for a 4-to-1 multiplexer with an active-low enable input can take the following form:

$$Y = \overline{EN}(\overline{S_1} \cdot \overline{S_0} \cdot D_0 + \overline{S_1} \cdot S_0 \cdot D_1 + S_1 \cdot \overline{S_0} \cdot D_2 + S_1 \cdot S_0 \cdot D_3) \qquad [3.4]$$

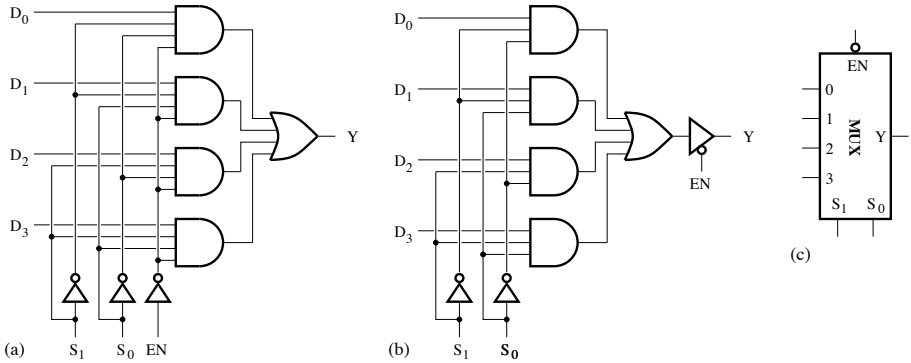


Figure 3.6. *4-to-1 multiplexer with an enable input: a) output set to zero when $EN = 1$; b) output set to high impedance state when $EN = 1$; c) symbol*

Equation [3.4] can equivalently be represented either by the truth table for the multiplexer of Figure 3.6(a) shown in Table 3.4, or by the truth table for the multiplexer of Figure 3.6(b) presented in Table 3.5.

EN	S_1	S_0	Y
1	x	x	0
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3

Table 3.4. *Truth table for the multiplexer shown in Figure 3.6(a)*

EN	S_1	S_0	Y
1	x	x	z
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3

Table 3.5. *Truth table for the multiplexer shown in Figure 3.6(b)*

An 8-to-1 multiplexer can be implemented either by using logic gates, as shown in Figure 3.7(a), or by connecting the 4-to-1 multiplexers, as illustrated in Figure 3.7(b). Its output is characterized by a logic equation that takes the following form:

$$\begin{aligned} Y = \overline{EN}(&\overline{S_2} \cdot \overline{S_1} \cdot \overline{S_0} \cdot D_0 + \overline{S_2} \cdot \overline{S_1} \cdot S_0 \cdot D_1 + \\ &\overline{S_2} \cdot S_1 \cdot \overline{S_0} \cdot D_2 + \overline{S_2} \cdot S_1 \cdot S_0 \cdot D_3 + S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot D_4 + \\ &S_2 \cdot \overline{S_1} \cdot S_0 \cdot D_5 + S_2 \cdot S_1 \cdot \overline{S_0} \cdot D_6 + S_2 \cdot S_1 \cdot S_0 \cdot D_7) \end{aligned} \quad [3.5]$$

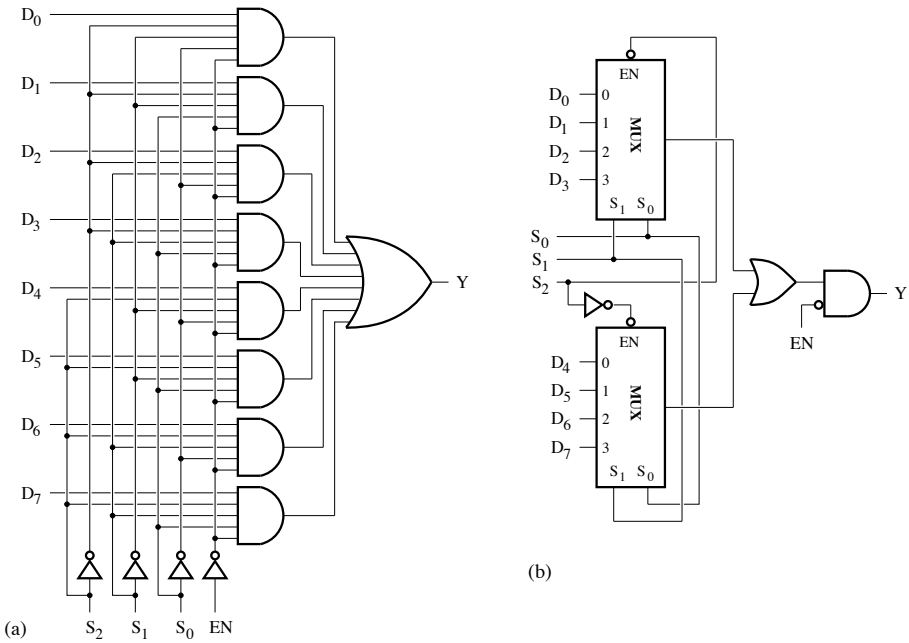


Figure 3.7. *8-to-1 multiplexer implemented using a) logic ports or b) 4-to-1 multiplexers*

Table 3.6 shows the truth table of the 8-to-1 multiplexer with an active-low enable input.

In general, a multiplexer with 2^n data inputs has n selection lines and a single output. The output takes the level of the data input whose number can be equal to the equivalent decimal value of the binary code applied at the selection inputs.

EN	S_0	S_1	S_0	Y
1	x	x	x	0
0	0	0	0	D_0
0	0	0	1	D_1
0	0	1	0	D_2
0	0	1	1	D_3
0	1	0	0	D_4
0	1	0	1	D_5
0	1	1	0	D_6
0	1	1	1	D_7

Table 3.6. *Truth table for an 8-to-1 multiplexer*

3.3. Demultiplexer and decoder

A decoder (DEC) is a logic circuit that activates only one of the outputs for each possible combination of input variables. It can thus be used to detect a binary code.

A demultiplexer (DMUX) is a logic circuit that switches a data input toward one of the outputs depending on the selection code.

The logic circuit and the symbol for the 1-out-of-2 decoder are illustrated in Figures 3.8(a) and 3.8(b), respectively. The logic equations for the output variables are given by:

$$Y_0 = EN \cdot \overline{D} \qquad [3.6]$$

and

$$Y_1 = EN \cdot D \qquad [3.7]$$

where D is the variable associated with the data input and EN represents the enable signal. Table 3.7 gives the truth table for the 1-out-of-2 decoder with an active-high input.

Decoders are most often designed with an active-low enable input, as can be seen in the logic circuit and the symbol given in Figures 3.9(a) and 3.9(b) for a 1-out-of-2 decoder. The logic equations for the output can be written as follows:

$$Y_0 = \overline{EN} \cdot \overline{D} \qquad [3.8]$$

and

$$Y_1 = \overline{EN} \cdot D \qquad [3.9]$$

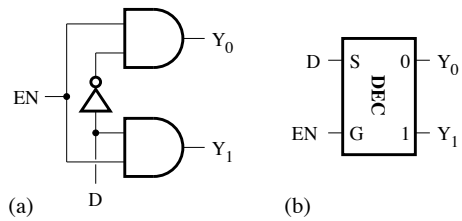


Figure 3.8. *1-out-of-2 decoder with an active-high enable input*

EN	D	Y_1	Y_0
0	x	0	0
1	0	0	1
1	1	1	0

Table 3.7. *Truth table for the decoder*

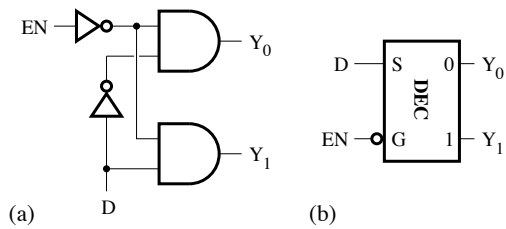


Figure 3.9. *1-out-of-2 decoder with an active-low enable input*

This makes it possible to construct the truth table given in Table 3.8.

Figure 3.10(a) represents the logic circuit of a 1-to-2 demultiplexer (or 1 : 2 demultiplexer). Figures 3.10(b) and 3.10(c) depict the symbols that are generally used to represent a 1-to-2 demultiplexer.

EN	D	Y_1	Y_0
1	x	0	0
0	0	0	1
0	1	1	0

Table 3.8. *Truth table for the decoder*

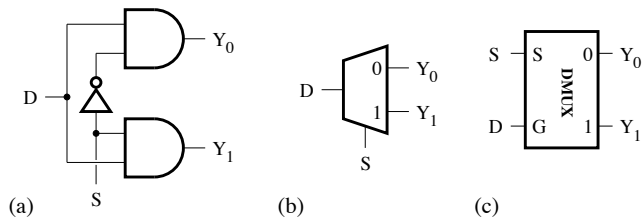


Figure 3.10. *1-to-2 demultiplexer: a) logic circuit, b) schematic diagram illustrating the working principle and c) symbols*

The $1:2$ demultiplexer is characterized by the following equations:

$$Y_0 = \overline{S} \cdot D \qquad [3.10]$$

$$Y_1 = S \cdot D \qquad [3.11]$$

The operation of the 1-to-2 demultiplexer is explained in the schematic diagram shown in Figure 3.11 and its truth table is represented by Table 3.9. The output Y_0 is selected when $S = 0$, and Y_1 when $S = 1$.

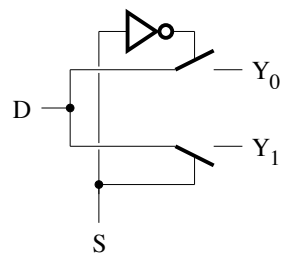


Figure 3.11. *Schematic diagram of a 1-to-2 demultiplexer*

It must be noted that to transform a decoder to a demultiplexer, it is enough to connect the data signal to the enable input *G*. Additionally, the input *G* is useful in establishing the connections required to associate several decoders in order to increase the length of the binary words that can be processed.

S	Y_0	Y_1
0	D	0
1	0	D

Table 3.9. *Truth table of a 1-to-2 demultiplexer*

The 1-to-2 demultiplexer shown in Figure 3.12 makes use of the fact that a three-state buffer is equivalent to an open or closed switched depending on the logic level applied to the selection input. The corresponding truth table is represented in Figure 3.10.

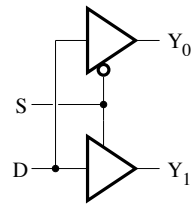


Figure 3.12. *Three-state buffer based 1-to-2 demultiplexer*

S	Y_1	Y_0
0	z	D_0
1	D_1	z

Table 3.10. *Truth table of the 1-to-2 demultiplexer*

Figure 3.13(a) depicts the logic circuit for a 2-out-of-4 decoder that is implemented using logic gates. A 2-out-of-4 decoder can also be implemented by connecting two 1-out-of-2 decoders as illustrated in Figure 3.13(b). The symbolic representation of a 2-out-of-4 decoder is given in Figure 3.13(c). The logic circuit of the 1-to-4 demultiplexer represented in Figure 3.14(a) uses logic gates, while the circuit shown in Figure 3.14(b) uses two 1-to-2 demultiplexers. Figure 3.14(c) depicts the symbol of a 1-to-4 demultiplexer. The logic equations obtained in each case are as follows:

– 2-out-of-4 decoder:

$$Y_0 = \overline{S_1} \cdot \overline{S_0}, \quad Y_1 = \overline{S_1} \cdot S_0, \quad Y_2 = S_1 \cdot \overline{S_0}, \quad \text{and} \quad Y_3 = S_1 \cdot S_0 \qquad [3.12]$$

– 1-to-4 demultiplexer:

$$Y_0 = \overline{S_1} \cdot \overline{S_0} \cdot D,\ Y_1 = \overline{S_1} \cdot S_0 \cdot D,\ Y_2 = S_1 \cdot \overline{S_0} \cdot D,\ \text{and}\ Y_3 = S_1 \cdot S_0 \cdot D \qquad [3.13]$$

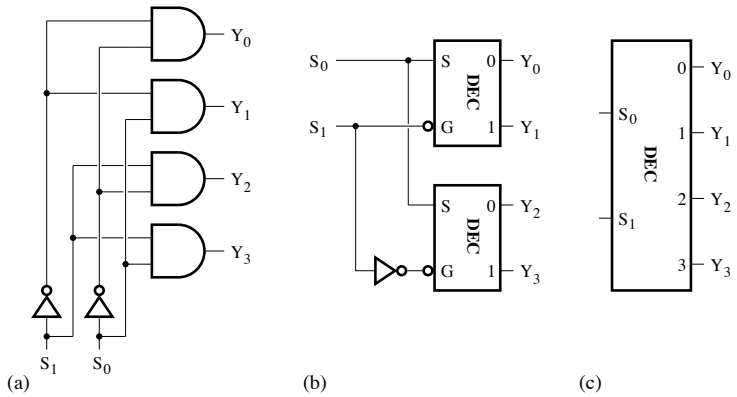


Figure 3.13. *2-out-of-4 decoder: implemented using a) logic gates or b) 1-out-2 decoders; c) symbol*

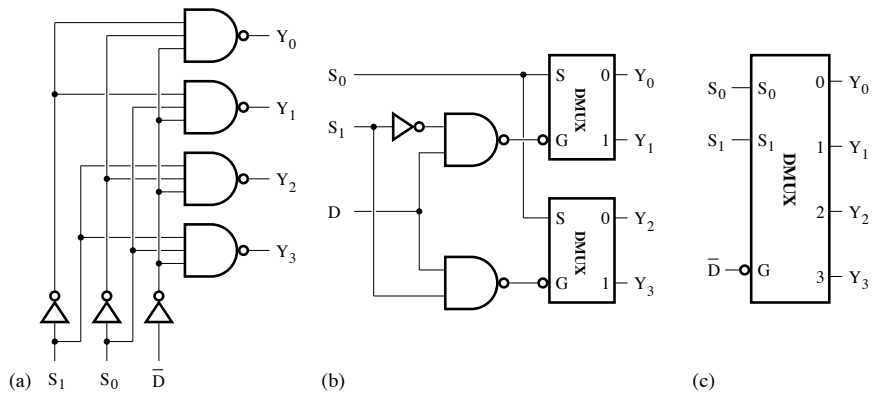


Figure 3.14. *1-to-4 demultiplexers: implemented using a) logic gates or b) 1-to-2 demultiplexers; c) symbol*

Truth tables of the 2-out-of-4 decoder and 1-to-4 demultiplexer are represented by Tables 3.11 and 3.12.

A 1-to-8 demultiplexer can be implemented using logic gates as shown in Figure 3.15(a) or using 1-to-4 multiplexers as illustrated in Figure 3.15(b). It can be represented by the symbol given in Figure 3.15(c). The output logic equations can be written as:

$$\begin{aligned} Y_0 &= \overline{S_2} \cdot \overline{S_1} \cdot \overline{S_0} \cdot D \quad Y_1 = \overline{S_2} \cdot \overline{S_1} \cdot S_0 \cdot D \quad Y_2 = \overline{S_2} \cdot S_1 \cdot \overline{S_0} \cdot D \\ Y_3 &= \overline{S_2} \cdot S_1 \cdot S_0 \cdot D \quad Y_4 = S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot D \quad Y_5 = S_2 \cdot \overline{S_1} \cdot S_0 \cdot D \\ Y_6 &= S_2 \cdot S_1 \cdot \overline{S_0} \cdot D \quad Y_7 = S_2 \cdot S_1 \cdot S_0 \cdot D \end{aligned} \qquad [3.14]$$

S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Table 3.11. *Truth table of a 2-out-of-4 decoder*

S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Table 3.12. *Truth table of a 1-to-4 demultiplexer*

Table 3.13 gives the truth table of the 1-to-8 demultiplexer. The output, to which the input data are transferred, is identified by a decimal number corresponding to the binary code applied to the selection inputs.

In general, a decoder is a logic circuit with n inputs and 2^n outputs, of which only one is active at any time. A demultiplexer has one data input, n select inputs, and 2^n outputs. Integrated decoders most often have one enable input. As the latter can also serve as a data input, it is possible to transform a decoder with an enable input to a demultiplexer.

Figure 3.16 shows a matrix-type structure that makes it possible to implement a 4-out-of-16 decoder using two 2-out-of-4 decoders and 16 AND gates. It offers the advantage of reducing the maximum number of inputs per logic gate and is used most often to construct memory networks. In general, to implement an n-out-of-2^n decoder, two decoders of the types, p-out-of-2^p and q-out-of-2^q, where $p + q = n$, and 2^n 2-input AND gates are required.

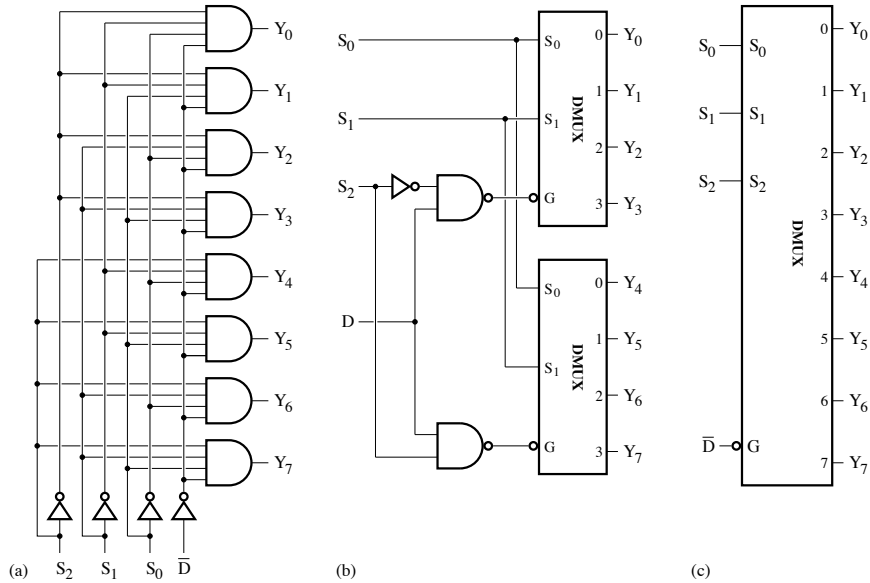


Figure 3.15. *1-to-8 demultiplexer implemented using a) logic gates or b) 1-to-4 demultiplexers; c) symbol*

S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	D
0	0	1	0	0	0	0	0	0	D	0
0	1	0	0	0	0	0	0	D	0	0
0	1	1	0	0	0	0	D	0	0	0
1	0	0	0	0	0	D	0	0	0	0
1	0	1	0	0	D	0	0	0	0	0
1	1	0	0	D	0	0	0	0	0	0
1	1	1	D	0	0	0	0	0	0	0

Table 3.13. *Truth table for an 1-to-8 demultiplexer*

3.4. Implementation of logic functions using multiplexers or decoders

A logic function can also be implemented using multiplexers or decoders.

3.4.1. *Multiplexer*

In general, a 2^{n-1}-to-1 multiplexer is required for the implementation of an n-variable logic function. A commonly adopted approach consists of linking $n-1$

variables to the selection inputs of the multiplexer and connecting the data inputs of the multiplexer to either the logic level 1 or 0, or to the remaining variable or its complement.

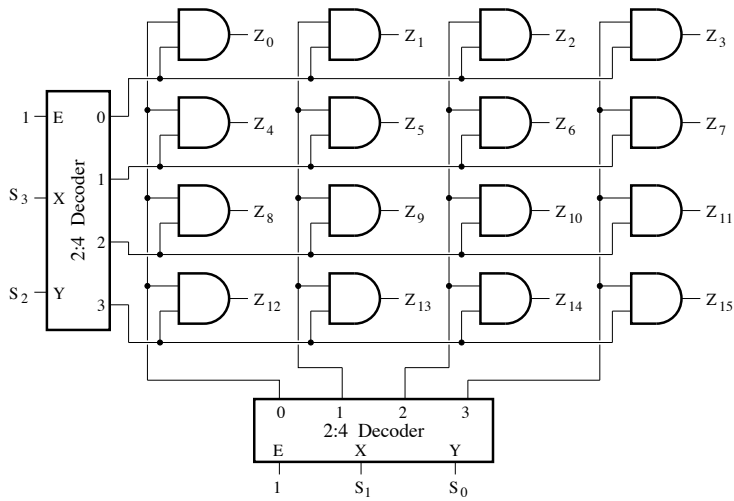


Figure 3.16. *2-out-of-16 decoder based on a matrix-type structure*

Implement the following 4-variable logic function using an 8-to-1 multiplexer:

$$Y(A, B, C, D) = \sum m(1, 2, 3, 4, 8, 12, 13, 15) \qquad [3.15]$$

Table 3.14 gives the truth table for the function Y. Considering the variables A, B and C, as the selection inputs to the multiplexer, it becomes possible to regroup the rows of the truth table in pairs, with each pair being characterized by the same combination of selection inputs. This translates to the factorization of 1, 0, D or $\overline{D}$, as illustrated in the truth table or the Karnaugh maps shown in Figure 3.17(a). The logic circuit of the 8-to-1 multiplexer configured to implement the function Y is given in Figure 3.17(b).

Using a multiplexer makes it possible to easily modify (or reconfigure) a logic function.

A	B	C	D	Y	
0	0	0	0	0	D
0	0	0	1	1	
0	0	1	0	1	1
0	0	1	1	1	
0	1	0	0	1	$\overline{D}$
0	1	0	1	0	
0	1	1	0	0	0
0	1	1	1	0	
1	0	0	0	1	$\overline{D}$
1	0	0	1	0	
1	0	1	0	0	0
1	0	1	1	0	
1	1	0	0	1	1
1	1	0	1	1	
1	1	1	0	0	D
1	1	1	1	1	

Table 3.14. *Truth table for the logic function Y*

3.4.2. *Decoder*

Even though a decoder is not considered as a universal component, it can still be used to implement logic functions. Any function with n variables may be implemented by an n-out-of-2^n decoder associated with an OR (or NAND) logic gate.

Use a 3-out-of-8 decoder and OR logic gates to implement the following logic functions:

$$P(A, B, C) = \sum m(0, 1, 3, 7) \quad [3.16]$$

and

$$Q(A, B, C) = \sum m(4, 6, 7) \quad [3.17]$$

As each of the functions is given in the canonical form, it is only necessary to generate the corresponding sum-of-products form, as illustrated in Figure 3.18.

The approach based on using a decoder and OR gates offers the advantage of allowing the implementation of several logic functions at the same time.

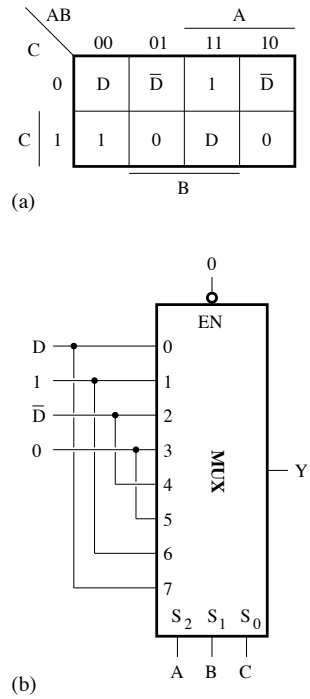


Figure 3.17. *a) Karnaugh map; b) logic circuit*

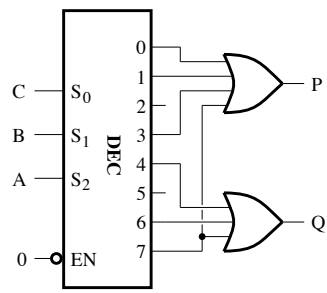


Figure 3.18. *Implementation of the functions P and Q using a 3-out-of-8 decoder*

3.5. Encoders

An encoder, in general, is a logic circuit that allows for the conversion of input information in a given code. It generally has more input variables than output variables.

3.5.1. *4:2 encoder*

A $4:2$ encoder, in its simplest form, is supposed to function with a single active input among four inputs and to generate a binary code through the two outputs. Consequently, there are only five permitted combinations of the input variables out of the possible 16. We can, thus, establish the truth tables shown in Tables 3.15 and 3.16.

D_3	D_2	D_1	D_0	Y_1	Y_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
...	...	...	...	x	x
⋮	⋮	⋮	⋮	⋮	⋮
...	...	...	...	x	x

Table 3.15. *Truth table (case 1)*

D_3	D_2	D_1	D_0	Y_1	Y_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
...	...	...	...	0	0
⋮	⋮	⋮	⋮	⋮	⋮
...	...	...	...	0	0

Table 3.16. *Truth table (case 2)*

– *Case 1*

In this case, the outputs Y_1 and Y_0 are considered to be don't care states for the 11 combinations of input variables, which are not explicitly defined in the truth table. The Karnaugh maps represented in Figures 3.19 and 3.20 allow the determination of the logic equations for Y_1 and Y_0, respectively. Figure 3.21 shows the logic circuit for the resulting $4:2$ encoder.

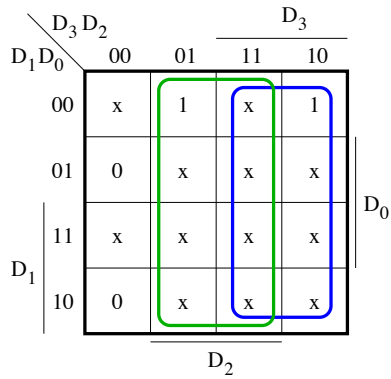


Figure 3.19. $Y_1 = D_3 + D_2$

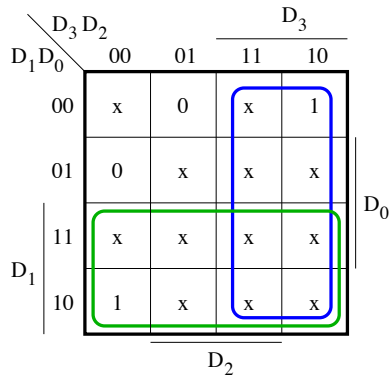


Figure 3.20. $Y_0 = D_3 + D_1$

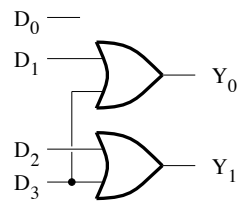


Figure 3.21. $4:2$ *encoder (case 1)*

– *Case 2*

In this case, the outputs Y_1 and Y_0 are assumed to take the logic level 0 for the 11 combinations of the input variables that are not explicitly defined in the truth table. The logic equations for Y_1 and Y_0 can be written as follows:

$$\begin{aligned} Y_1 &= \overline{D_3} \cdot D_2 \cdot \overline{D_1} \cdot \overline{D_0} + D_3 \cdot \overline{D_2} \cdot \overline{D_1} \cdot \overline{D_0} \\ &= \overline{D_1} \cdot \overline{D_0}(D_3 \oplus D_2) \end{aligned} \quad [3.18]$$

and

$$\begin{aligned} Y_0 &= \overline{D_3} \cdot \overline{D_2} \cdot D_1 \cdot \overline{D_0} + D_3 \cdot \overline{D_2} \cdot \overline{D_1} \cdot \overline{D_)} \\ &= \overline{D_2} \cdot \overline{D_0}(D_3 \oplus D_1) \end{aligned} \quad [3.19]$$

The logic circuit for the resulting $4:2$ encoder is illustrated in Figure 3.22.

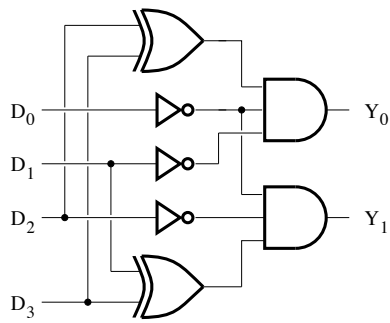


Figure 3.22. $4:2$ *encoder (case 2)*

NOTE 3.1.– The encoder generates the output $Y_1Y_0 = 00$ if the input D_0 is set at either 1 or 0. The addition of a validation output V makes it possible to distinguish between these two cases.

Table 3.17 shows the truth table of a $4:2$ encoder with a validation output. The logic equation for the validation output is given by:

$$V = D_3 + D_2 + D_1 + D_0 \quad [3.20]$$

The logic circuit obtained for the $4:2$ encoder with a validation output is represented in Figure 3.23.

D_3	D_2	D_1	D_0	Y_1	Y_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1
...	...	...	...	x	x	x
⋮	⋮	⋮	⋮	⋮	⋮	⋮
...	...	...	...	x	x	x

Table 3.17. *Truth table of the* $4:2$ *encoder with a validation output*

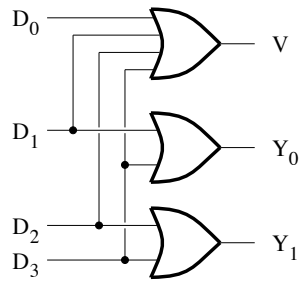


Figure 3.23. $4:2$ *encoder with a validation output*

3.5.2. *8:3 encoder*

An $8:3$ encoder with a validation output generates a unique 4-bit sequence as the output for each combination of input variables with a single input set at 1. Among the 256 possible input combinations, there are only nine permitted combinations. We can construct the truth table as shown in Table 3.18.

As don't care states can be used to minimize the logic equation for each output, analyzing the truth table can help us deduce that the simplest expression corresponds to an OR function for input variables taking the logic level 1 at the same time as the output of interest. In this way, we obtain the following logic equations:

$$Y_2 = D_7 + D_6 + D_5 + D_4 \quad [3.21]$$

$$Y_1 = D_7 + D_6 + D_3 + D_2 \quad [3.22]$$

$$Y_0 = D_7 + D_5 + D_3 + D_1 \quad [3.23]$$

and

$$V = D_7 + D_6 + D_5 + D_4 + D_3 + D_2 + D_1 + D_0 \qquad [3.24]$$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_2	Y_1	Y_0	V
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	1	0	0	0	1	0	1
0	0	0	0	1	0	0	0	0	1	1	1
0	0	0	1	0	0	0	0	1	0	0	1
0	0	1	0	0	0	0	0	1	0	1	1
0	1	0	0	0	0	0	0	1	1	0	1
1	0	0	0	0	0	0	0	1	1	1	1
...	...	...	...	...	...	...	...	x	x	x	x
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
...	...	...	...	...	...	...	...	x	x	x	x

Table 3.18. *Truth table for an* $8:3$ *encoder*

Figure 3.24 shows the logic circuit for an $8:3$ encoder with a validation output. The change in the logic level of the input D_0 is only detected through the validation output.

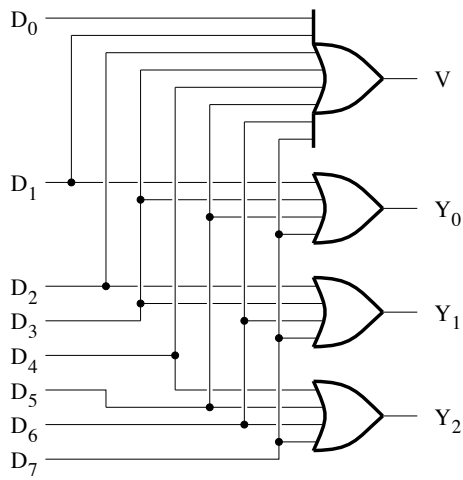


Figure 3.24. $8:3$ *encoder with a validation output*

3.5.3. *Priority encoder*

A $2^n : n$ encoder only operates correctly if none of the inputs or one single input is at logic level 1. When more than one input simultaneously takes logical level 1, the coding is wrong. One solution, in this case, consists of using a priority encoder.

A priority encoder generates a binary code corresponding to the number of the active input with highest priority (or, most often, the highest number). It can be used in the following applications:

– keyboard encoder: when several keys are pressed simultaneously, only the key with the highest number is taken into consideration;

– unit processing interrupt requests in a microprocessor: in case of simultaneous interrupt requests, only the request with the highest priority is accepted.

3.5.3.1. *4:2 priority encoder*

The truth table for a $4 : 2$ priority encoder is given in Table 3.19, where *x* represents a don't care state. For an input word, the active bit with the highest weight has priority.

D_3	D_2	D_1	D_0	Y_1	Y_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Table 3.19. *Truth table for a 4:2 priority encoder*

The Karnaugh maps shown in Figures 3.25–3.27 are constructed assuming that each indifferent state can take the logic level 0 or logic level 1. The resulting logic equations can be written as follows:

$$Y_1 = D_3 + D_2 \qquad [3.25]$$

$$Y_0 = D_3 + \overline{D_2} \cdot D_1 \qquad [3.26]$$

and

$$V = D_3 + D_2 + D_1 + D_0 \qquad [3.27]$$

The logic circuit of the $4 : 2$ priority encoder can then be realized as illustrated in Figure 3.28.

3.5.3.2. *4:2 priority encoders with cascading capability*

The priority encoders offered by manufacturers of integrated circuits have additional inputs and outputs (*enable-in input*, EI, *enable-out output*, E0, *group signal output*, GS or V) that can be required for cascade connections.

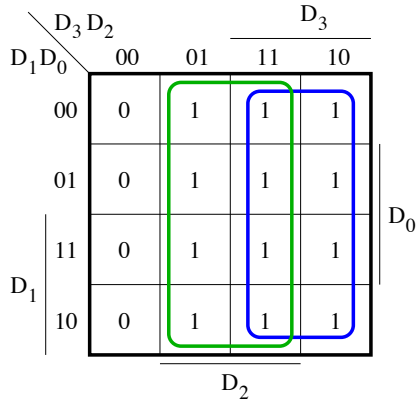


Figure 3.25. *Representation of* $Y_1 = D_3 + D_2$

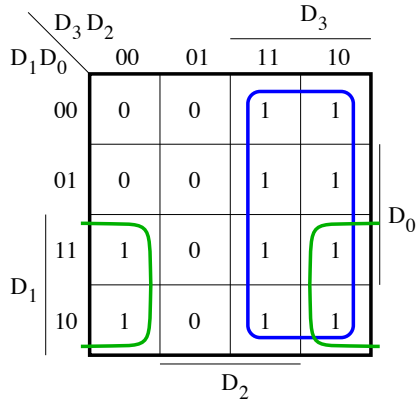


Figure 3.26. *Representation of* $Y_0 = D_3 + \overline{D_2} D_1$

To design a $4 : 2$ priority encoder with cascading capability, we begin by constructing the truth table as illustrated in Table 3.20. As the encoder is based on active low logic, the output V indicates when EI takes the logic level 0 and a single entry among D_k $(k = 0, 1, 2, 3)$ takes logic level 1. Karnaugh maps shown in

Figures 3.29–3.32 are constructed and then used to determine the logic equations of the outputs as follows:

$$Y_1 = (D_3 + D_2)\overline{EI} \quad [3.28]$$

$$Y_0 = (D_3 + \overline{D_2} \cdot D_1)\overline{EI} \quad [3.29]$$

$$V = (D_3 + D_2 + D_1 + D_0)EI = \overline{E0} \cdot \overline{EI} \quad [3.30]$$

and

$$E0 = \overline{D_3} \cdot \overline{D_2} \cdot \overline{D_1} \cdot \overline{D_0} \cdot \overline{EI} \quad [3.31]$$

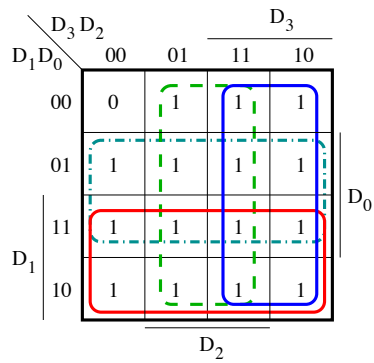


Figure 3.27. *Representation of* $V = D_3 + D_2 + D_1 + D_0$

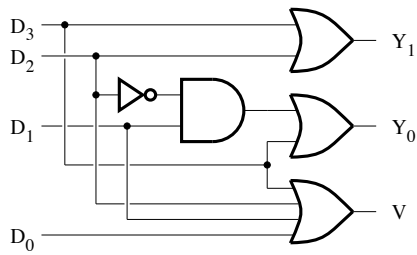


Figure 3.28. $4:2$ *priority encoder with a validation output*

The transcription of these equations using logic gates results in the circuit shown in Figure 3.33(a). Figure 3.33(b) depicts the symbol for a $4:2$ priority encoder with cascading capability. The $8:3$ priority encoder shown in Figure 3.33(c) is

implemented by cascading two $4 : 2$ priority encoders. This brings into play the following logic expressions:

$$E0 = E0_1 \quad [3.32]$$

and

$$EI_1 = E0_2 = \overline{D_7} \cdot \overline{D_6} \cdot \overline{D_5} \cdot \overline{D_4} \cdot \overline{EI} \quad [3.33]$$

where $EI = EI_2$. Table 3.21 shows the truth table for the $8 : 3$ priority encoder.

EI	D_3	D_2	D_1	D_0	Y_1	Y_0	V	$E0$
0	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	1	0
0	0	0	1	x	0	1	1	0
0	0	1	x	x	1	0	1	0
0	1	x	x	x	1	1	1	0
1	x	x	x	x	0	0	0	0

Table 3.20. *Truth table for $4 : 2$ priority encoder with cascading capability*

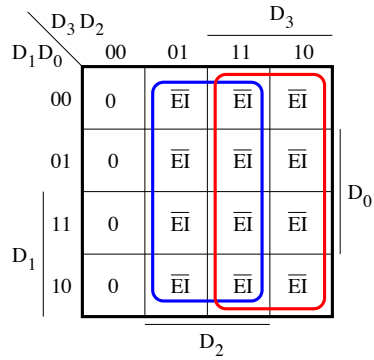


Figure 3.29. *Representation of $Y_1 = (D_3 + D_2)\overline{EI}$*

Devices such as $8 : 3$ priority encoders with cascading capability and $10 : 4$ priority encoders are available as commercial integrated circuits. They are especially useful for applications that use *binary coded decimal* (BCD) representation, coding of keyboard keys and selection of a numerical range.

3.5.3.3. *10:4 priority encoder*

A $10 : 4$ priority encoder or decimal binary priority encoder carries out coding of input logic levels such that only a change in the logic level of the highest ranked input is taken into consideration.

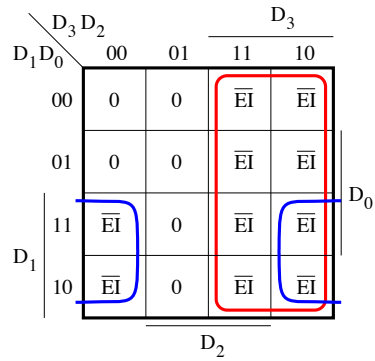


Figure 3.30. *Representation of* $Y_0 = (D_3 + \overline{D_2} \cdot D_1)\overline{EI}$

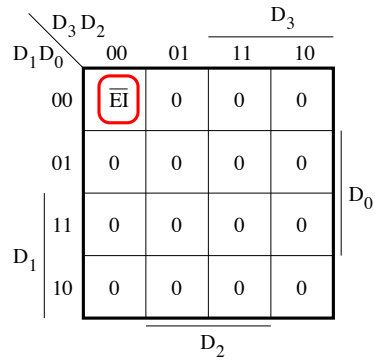


Figure 3.31. *Representation of* $E0 = \overline{D_3} \cdot \overline{D_2} \cdot \overline{D_1} \cdot \overline{D_0} \cdot \overline{EI}$

As the decimal zero or D_0 corresponds to the case where all the inputs are at the low logic level, the input D_0 is omitted. The input D_i, with i being a number between 1 and 9, can only activate the output Y_j ($j = 0, 1, 2, 3$) if no input with a higher priority and other than those that also activate Y_j takes the high logic level.

Based on the conversion table shown in Table 3.22, it is possible to formulate a logic proposition for each output. Thus, the output Y_0 takes the high logic level when one of the following is true:

– D_1 is at high logic level, and D_2, D_4, D_6 and D_8 are at low logic level;

– D_3 is at high logic level, and D_4, D_6 and D_8 are at low logic level;

– D_5 is at high logic level, and D_6 and D_8 are at low logic level;

EI	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_2	Y_1	Y_0	V	$E0$
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	x	0	0	1	1	0
0	0	0	0	0	0	1	x	x	0	1	0	1	0
0	0	0	0	0	1	x	x	x	0	1	1	1	0
0	0	0	0	1	x	x	x	x	1	0	0	1	0
0	0	0	1	x	x	x	x	x	1	0	1	1	0
0	0	1	x	x	x	x	x	x	1	1	0	1	0
0	1	x	x	x	x	x	x	x	1	1	1	1	0
1	x	x	x	x	x	x	x	x	0	0	0	0	0

Table 3.21. *Truth table for the* $8:3$ *priority encoder*

– D_7 is at high logic level, and D_8 is at low logic level;

– D_7 is at high logic level.

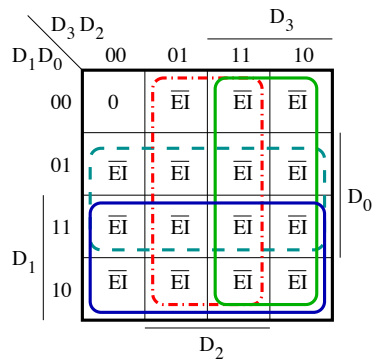


Figure 3.32. *Representation of* $V = (D_3 + D_2 + D_1 + D_0)\overline{EI}$

This translates to:

$$Y_0 = D_1 \cdot \overline{D_2} \cdot \overline{D_4} \cdot \overline{D_6} \cdot \overline{D_8} + D_3 \cdot \overline{D_4} \cdot \overline{D_6} \cdot \overline{D_8} + D_5 \cdot \overline{D_6} \cdot \overline{D_8} + D_7 \cdot \overline{D_8} + D_9 \quad [3.34]$$

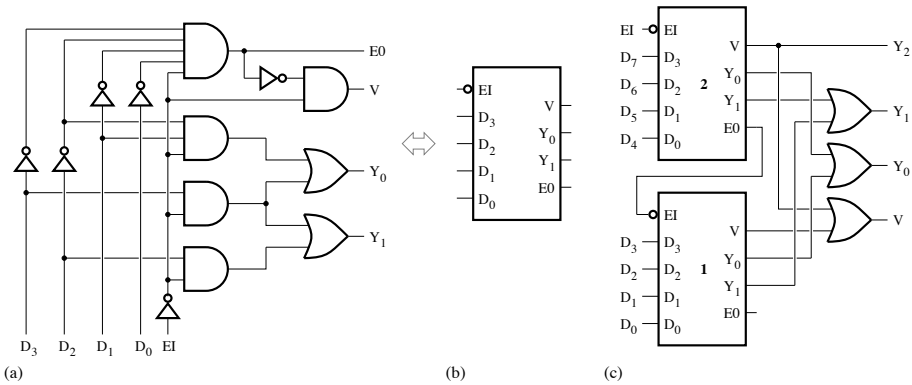


Figure 3.33. *Circuit a) and symbol b) for a* $4:2$ *priority encoder with cascading capability; c)* $8:3$ *priority encoder*

	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Table 3.22. *Conversion of decimal numbers from 0 to 9 into binary representation*

Similarly, the output Y_1 takes high logic level in one of the following cases:

– D_2 is at high logic level, and D_4, D_5, D_8 and D_9 are at low logic level;

– D_3 is at high logic level, and D_4, D_5, D_8 and D_9 are at low logic level;

– D_6 is at high logic level, and D_8 and D_9 are at low logic level;

– D_7 is at high logic level, and D_8 and D_9 are at low logic level.

This results in the following logic equation:

$$Y_1 = D_2 \cdot \overline{D_4} \cdot \overline{D_5} \cdot \overline{D_8} \cdot \overline{D_9} + D_3 \cdot \overline{D_4} \cdot \overline{D_5} \cdot \overline{D_8} \cdot \overline{D_9} + D_6 \cdot \overline{D_8} \cdot \overline{D_9} + D_7 \cdot \overline{D_8} \cdot \overline{D_9} \quad [3.35]$$

The output Y_2 takes high logic level in one of the following cases:

– D_4 is at high logic level, and D_8 and D_9 are at low logic level;

– D_5 is at high logic level, and D_8 and D_9 are at low logic level;

– D_6 is at high logic level, and D_8 and D_9 are at low logic level;

– D_7 is at high logic level, and D_8 and D_9 are at low logic level.

This leads to the following equation:

$$Y_2 = D_4 \cdot \overline{D_8} \cdot \overline{D_9} + D_5 \cdot \overline{D_8} \cdot \overline{D_9} + D_6 \cdot \overline{D_8} \cdot \overline{D_9} + D_7 \cdot \overline{D_8} \cdot \overline{D_9} \quad [3.36]$$

Finally, the output Y_3 is at high logic level if D_8 is at high level or if D_9 is at high level. The resulting logic equation is, thus, given by:

$$Y_3 = D_8 + D_9 \quad [3.37]$$

Figure 3.34 depicts the logic circuit for the priority encoder 74LS147. For this kind of circuit, the inputs and outputs are active at the low logic level. The truth table of Table 3.23 shows that the priority of each input is determined by its rank in decimal. Each active input is only taken into consideration if, and only if, all entries of a higher rank are inactive.

3.6. Transcoders

In addition to the encoder and the decoder, we can also distinguish the transcoder that allows for the conversion of a given code to a different code.

3.6.1. *Binary code and Gray code*

Gray code (or reflected binary code) is used in angle or positional sensors and also in applications where the likelihood of commutation errors is to be reduced.

Gray code is a code constructed such that the representation of two consecutive numbers only differs by a single bit.

Table 3.24 shows the 4-bit binary Gray code conversion (or for the numbers from 0 to 15).

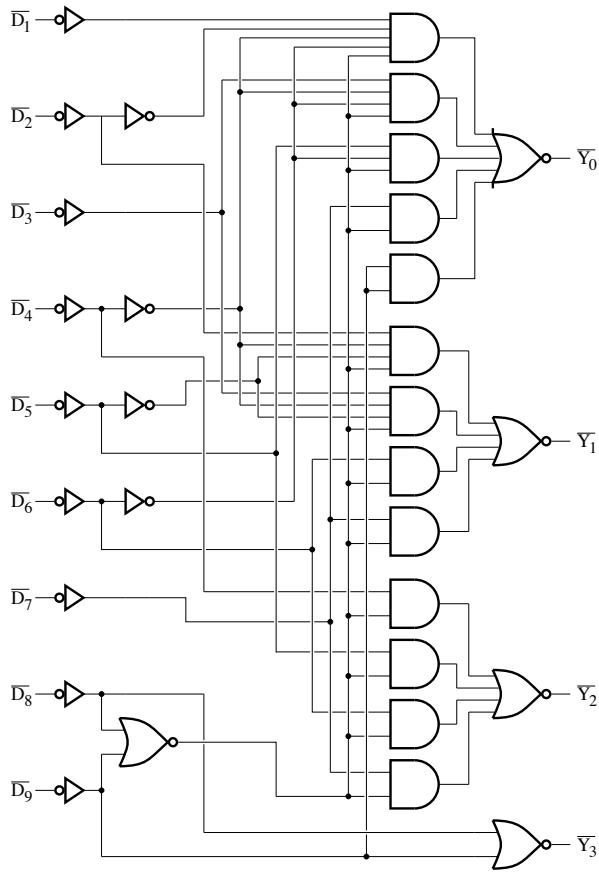


Figure 3.34. *10:4 priority encoder (integrated circuit 74LS147)*

3.6.1.1. *Binary to Gray code converter*

Logic equations associated with the Gray code bits can be determined by observing that the bits G_3 and B_3 are identical and using Karnaugh maps represented in Figures 3.35–3.37 in the case of G_2, G_1 and G_0 bits, respectively.

Thus, the binary to Gray code converter is characterized by:

$$G_3 = B_3 \quad [3.38]$$

$$G_2 = B_3 \oplus B_2 \quad [3.39]$$

$$G_1 = B_2 \oplus B_1 \quad [3.40]$$

$$G_0 = B_1 \oplus B_0 \quad [3.41]$$

where $\oplus$ represents the exclusive OR function. The corresponding logic circuit is illustrated in Figure 3.38.

$\overline{D_1}$	$\overline{D_2}$	$\overline{D_3}$	$\overline{D_4}$	$\overline{D_5}$	$\overline{D_6}$	$\overline{D_7}$	$\overline{D_8}$	$\overline{D_9}$	$\overline{Y_3}$	$\overline{Y_2}$	$\overline{Y_1}$	$\overline{Y_0}$
1	1	1	1	1	1	1	1	1	1	1	1	1
x	x	x	x	x	x	x	x	0	0	1	1	0
x	x	x	x	x	x	x	0	1	0	1	1	1
x	x	x	x	x	x	0	1	1	1	0	0	0
x	x	x	x	x	0	1	1	1	1	0	0	1
x	x	x	x	0	1	1	1	1	1	0	1	0
x	x	x	0	1	1	1	1	1	1	0	1	1
x	x	0	1	1	1	1	1	1	1	1	0	0
x	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

Table 3.23. *Truth table for the priority encoder 74LS147*

Decimal	Binary code				Gray code			
Number	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Table 3.24. *Binary and Gray code for numbers from 0 to 15*

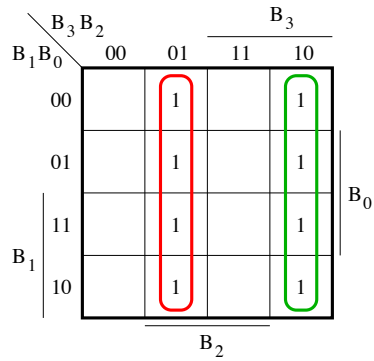


Figure 3.35. *Representation of* $G_2 = B_3 \cdot \overline{B_2} + \overline{B_3} \cdot B_2 = B_3 \oplus B_2$

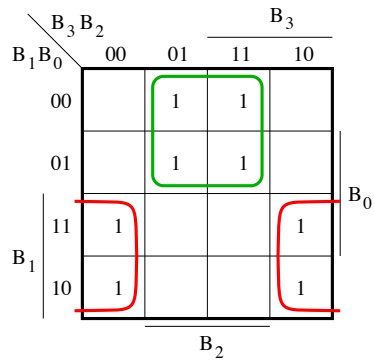


Figure 3.36. *Representation of* $G_1 = B_2 \cdot \overline{B_1} + \overline{B_2} \cdot B_1 = B_2 \oplus B_1$

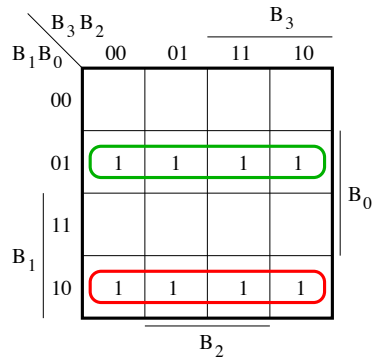


Figure 3.37. *Representation of* $G_0 = B_1 \cdot \overline{B_0} + \overline{B_1} \cdot B_0 = B_1 \oplus B_0$

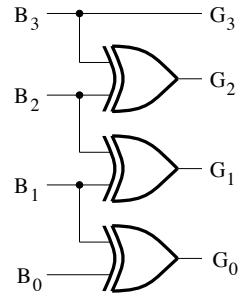


Figure 3.38. *Binary to Gray code converter*

3.6.1.2. *Gray code to binary converter*

Logic equations for the Gray code to binary code converter can be deduced from the conversion table given in Table 3.24. As the B_3 and G_3 bits are identical, the construction of Karnaugh maps is only required for each of the following bits: B_2, B_1 and B_0 (see Figures 3.39–3.41). For the maps shown in Figures 3.40 and 3.41, we have:

$$\begin{aligned} B_1 &= \overline{G_3} \cdot \overline{G_2} \cdot G_1 + \overline{G_3} \cdot G_2 \cdot \overline{G_1} + G_3 \cdot G_2 \cdot G_1 + G_3 \cdot \overline{G_2} \cdot \overline{G_1} \\ &= \overline{G_3}(G_2 \oplus G_1) + G_3(\overline{G_2 \oplus G_1}) \\ &= G_3 \oplus G_2 \oplus G_1 \end{aligned} \quad [3.42]$$

and

$$\begin{aligned} B_0 &= \overline{G_3} \cdot G_2 \cdot \overline{G_1} \cdot \overline{G_0} + G_3 \cdot \overline{G_2} \cdot \overline{G_1} \cdot \overline{G_0} + \\ &\qquad \overline{G_3} \cdot \overline{G_2} \cdot \overline{G_1} \cdot G_0 + G_3 \cdot G_2 \cdot \overline{G_1} \cdot G_0 + \overline{G_3} \cdot G_2 \cdot G_1 \cdot G_0 + \\ &\qquad G_3 \cdot \overline{G_2} \cdot G_1 \cdot G_0 + \overline{G_3} \cdot \overline{G_2} \cdot G_1 \cdot \overline{G_0} + G_3 \cdot G_2 \cdot G_1 \cdot \overline{G_0} \\ &= (G_3 \oplus G_2)\overline{G_1} \cdot \overline{G_0} + \\ &\qquad (\overline{G_3 \oplus G_2})\overline{G_1} \cdot G_0 + (G_3 \oplus G_2)G_1 \cdot G_0 + (\overline{G_3 \oplus G_2})G_1 \cdot \overline{G_0} \\ &= (G_3 \oplus G_2)(\overline{G_1 \oplus G_0}) + (\overline{G_3 \oplus G_2})(G_1 \oplus G_0) \\ &= G_3 \oplus G_2 \oplus G_1 \oplus G_0 \end{aligned} \quad [3.43]$$

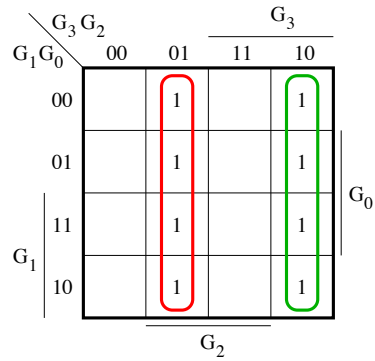


Figure 3.39. *Representation of* $B_2 = G_3 \cdot \overline{G_2} + \overline{G_3} \cdot G_2 = G_3 \oplus G_2$

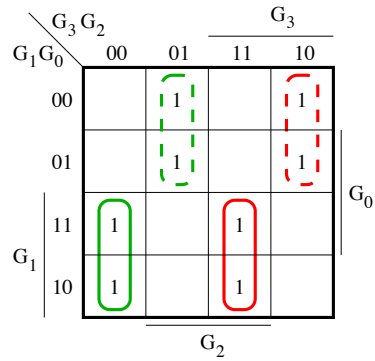


Figure 3.40. *Representation of* $B_1 = G_3 \oplus G_2 \oplus G_1$

The logic equations for the Gray code to binary code convertor are thus of the form:

$$B_3 = G_3 \quad [3.44]$$

$$B_2 = G_3 \oplus G_2 \quad [3.45]$$

$$B_1 = G_3 \oplus G_2 \oplus G_1 \quad [3.46]$$

$$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0 \quad [3.47]$$

Using exclusive OR gates, the logic function illustrated in Figure 3.42 can be implemented.

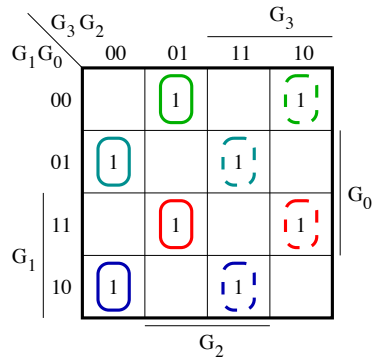


Figure 3.41. *Representation of* $B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$

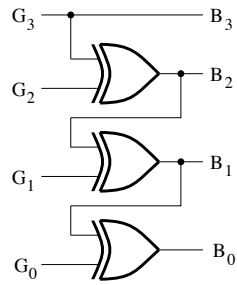


Figure 3.42. *Gray code to binary code converter*

3.6.2. *BCD and excess-3 code*

BCD corresponds to the representation in natural binary (here, with four bits of weights 8, 4, 2 and 1) of each digit of a decimal number. It is used, for example, to allow the content of a counter to be displayed.

The Excess-3 code (XS-3) is obtained by adding 3 to the decimal number to be converted before representing it in BCD form. It is considered to be a complementary BCD code as it also allows for the representation of both positive and negative numbers.

3.6.2.1. *BCD to XS-3 converter*

Table 3.25 gives the conversion table for converting BCD to XS-3. As the conversion only involves numbers from 0 to 9, the binary combinations associated

with the numbers from 10 to 15 can be considered to be don't care terms. Based on the conversion tables, Karnaugh maps of Figures 3.43–3.46 can be constructed to determine the logic equations for the outputs X_3, X_2, X_1, and X_0, respectively.

Decimal	BCD code				XS-3 code			
number	B_3	B_2	B_1	B_0	X_3	X_2	X_1	X_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
	1	0	1	0	x	x	x	x
	1	0	1	1	x	x	x	x
	1	1	0	0	x	x	x	x
	1	1	0	1	x	x	x	x
	1	1	1	0	x	x	x	x
	1	1	1	1	x	x	x	x

Table 3.25. *BCD to XS-3 conversion table*

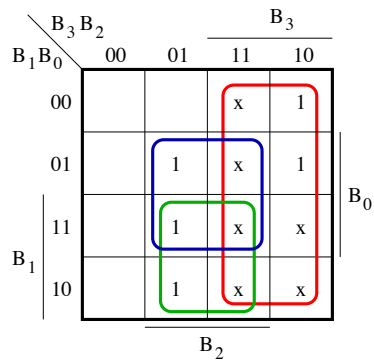


Figure 3.43. *Representation of* $X_3 = B_3 + B_2 \cdot B_1 + B_2 \cdot B_0$

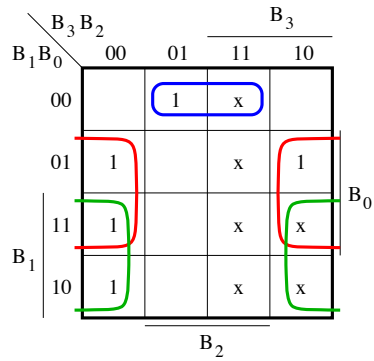


Figure 3.44. *Representation of* $X_2 = \overline{B_2} \cdot B_1 + \overline{B_2} \cdot B_0 + B_2 \cdot \overline{B_1} \cdot \overline{B_0}$

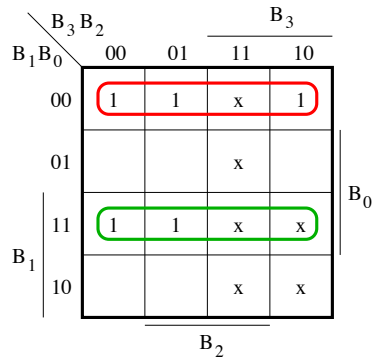


Figure 3.45. *Representation of* $X_1 = \overline{B_1} \cdot \overline{B_0} + B_1 \cdot B_0$

We thus obtain:

$$X_3 = B_3 + B_2 \cdot B_1 + B_2 \cdot B_0$$
$$= B_3 + B_2(B_1 + B_0) \quad [3.48]$$
$$X_2 = \overline{B_2} \cdot B_1 + \overline{B_2} \cdot B_0 + B_2 \cdot \overline{B_1} \cdot \overline{B_0}$$
$$= B_2 \oplus (B_1 + B_0) \quad [3.49]$$
$$X_1 = \overline{B_1} \cdot \overline{B_0} + B_1 \cdot B_0$$
$$= B_1 \oplus \overline{B_0} \quad [3.50]$$

and:

$$X_0 = \overline{B_0} \qquad [3.51]$$

The logic circuit for a BCD to XS-3 converter can then be realized as illustrated in Figure 3.47.

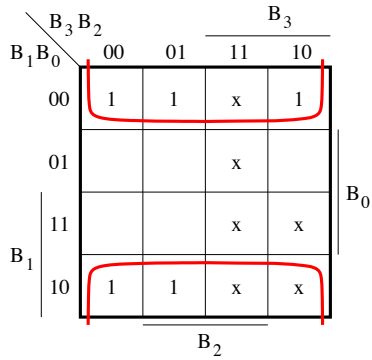


Figure 3.46. *Representation of $X_0 = \overline{B_0}$*

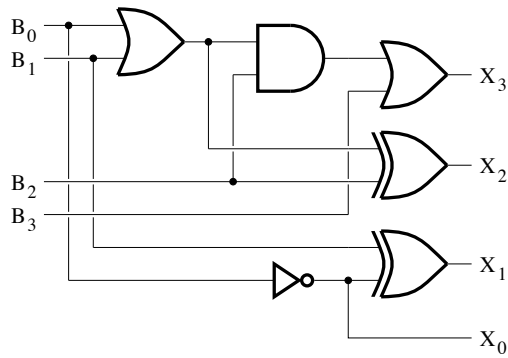


Figure 3.47. *BCD to XS-3 converter*

3.6.2.2. *XS-3 to BCD converter*

The XS-3 to BCD conversion table is given in Table 3.26. Among the 16 binary combinations that can be applied to the inputs (X_3, X_2, X_1, and X_0), only those associated with the numbers 0 to 9 are used and the others are considered as don't

care terms. The logic equations for the outputs (B_3, B_2, B_1, and B_0) are obtained by covering groups of adjacent cells in Karnaugh maps of Figures 3.48–3.51. We thus have:

$$\begin{aligned} B_3 &= X_3 \cdot X_2 + X_3 \cdot X_1 \cdot X_0 \\ &= X_3(X_2 + X_1 \cdot X_0) \end{aligned} \quad [3.52]$$

$$\begin{aligned} B_2 &= \overline{X_2} \cdot \overline{X_0} + \overline{X_2} \cdot \overline{X_1} + X_2 \cdot X_1 \cdot X_0 \\ &= X_2 \oplus (\overline{X_1 \cdot X_0}) = \overline{X_2 \oplus (X_1 \cdot X_0)} \end{aligned} \quad [3.53]$$

$$\begin{aligned} B_1 &= X_1 \cdot \overline{X_0} + \overline{X_1} \cdot X_0 \\ &= X_1 \oplus X_0 \end{aligned} \quad [3.54]$$

and

$$B_0 = \overline{X_0} \quad [3.55]$$

Decimal	XS-3 code				BCD code			
number	X_3	X_2	X_1	X_0	B_3	B_2	B_1	B_0
0	0	0	1	1	0	0	0	0
1	0	1	0	0	0	0	0	1
2	0	1	0	1	0	0	1	0
3	0	1	1	0	0	0	1	1
4	0	1	1	1	0	1	0	0
5	1	0	0	0	0	1	0	1
6	1	0	0	1	0	1	1	0
7	1	0	1	0	0	1	1	1
8	1	0	1	1	1	0	0	0
9	1	1	0	0	1	0	0	1
	1	1	0	1	x	x	x	x
	1	1	1	0	x	x	x	x
	1	1	1	1	x	x	x	x
	0	0	0	0	x	x	x	x
	0	0	0	1	x	x	x	x
	0	0	1	0	x	x	x	x

Table 3.26. *BCD to XS-3 conversion table*

Using logic gates, the logic circuit of the XS-3 to BCD converter is realized as shown in Figure 3.52.

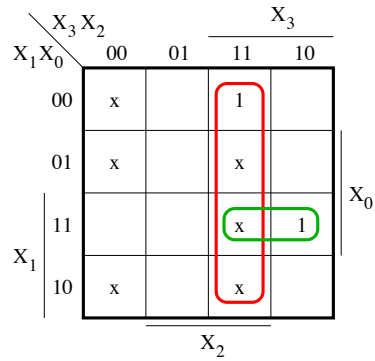


Figure 3.48. *Representation of* $B_3 = X_3 \cdot X_2 + X_3 \cdot X_1 \cdot X_0$

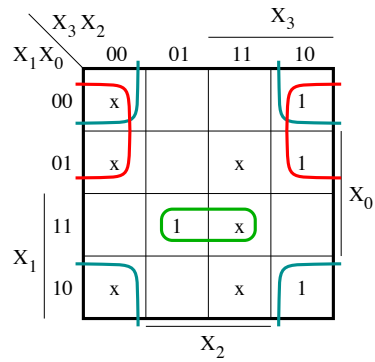


Figure 3.49. *Representation of* $B_2 = \overline{X_2} \cdot \overline{X_0} + \overline{X_2} \cdot \overline{X_1} + X_2 \cdot X_1 \cdot X_0$

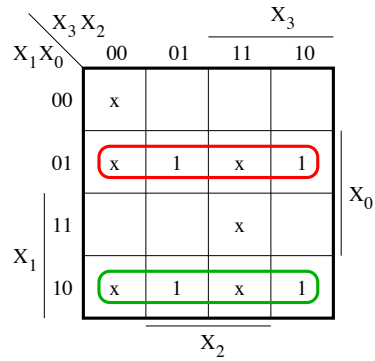


Figure 3.50. *Representation of* $B_1 = X_1 \cdot \overline{X_0} + \overline{X_1} \cdot X_0$

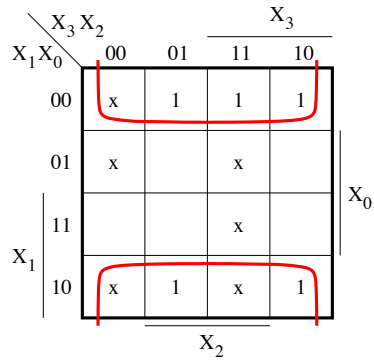


Figure 3.51. *Representation of* $B_0 = \overline{X_0}$

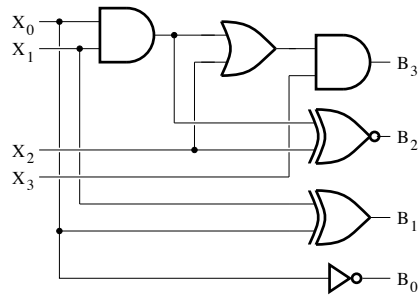


Figure 3.52. *XS-3 to BCD converter*

3.7. Parity check generator

Parity bits are bits that are added to data to be transmitted for error verification purposes. There are two types of parity: even and odd. The output of an even (odd) parity generator is set either to 1 (0) if the number of bits set at the high logic level or 1 in the input word is odd, or to 0 (1) if this number is even.

EXAMPLE 3.1.– The concatenation of a parity bit with a given piece of data results in a word that can have an odd number of bits at logic level 1 (or an even parity) or an even number of bits at logic level 1 (or an odd parity). Hence, to generate the parity, the number of bits at logic level 1 takes into account the parity bit, while for checking the parity, the parity bit is excluded from the total number of bits at logic level 1 and is only used to identify the type of parity. Table 3.27 presents three 8-bit words with the corresponding parity bits.

Word with	Parity	
8 bits	Even	Odd
00000000	0	1
00101000	0	1
01001100	1	0

Table 3.27. *Example of three 8-bit words with parity bits*

Parity is used in series communication systems and memories to detect transmission errors due to noise. After errors resulting in the modification of the logic level of the parity bit are detected by a checker, it is possible to envisage a correction by the retransmission of the data.

Table 3.28 gives the truth table of a parity generator for 4-bit words. The input data are of the form $D_3D_2D_1D_0$, and the output variable is either P_E (even parity) or P_O (odd parity). The Karnaugh map constructed based on the truth table, as shown in Figure 3.53, allows for the determination of the logic expression for P_E. We thus have:

$$\begin{aligned} P_E = {} & \overline{D_3} \cdot D_2 \cdot \overline{D_1} \cdot \overline{D_0} + D_3 \cdot \overline{D_2} \cdot \overline{D_1} \cdot \overline{D_0} \\ & + \overline{D_3} \cdot \overline{D_2} \cdot \overline{D_1} \cdot D_0 + D_3 \cdot D_2 \cdot \overline{D_1} \cdot D_0 + \overline{D_3} \cdot D_2 \cdot D_1 \cdot D_0 + \\ & + D_3 \cdot \overline{D_2} \cdot D_1 \cdot D_0 + \overline{D_3} \cdot \overline{D_2} \cdot D_1 \cdot \overline{D_0} + D_3 \cdot D_2 \cdot D_1 \cdot \overline{D_0} \end{aligned} \quad [3.56]$$

or equivalently:

$$\begin{aligned} P_E = {} & (\overline{D_3} \cdot D_2 + D_3 \cdot \overline{D_2})\overline{D_1} \cdot \overline{D_0} + (\overline{D_3} \cdot \overline{D_2} + D_3 \cdot D_2)\overline{D_1} \cdot D_0 \\ & + (\overline{D_3} \cdot D_2 + D_3 \cdot \overline{D_2})D_1 \cdot D_0 + (\overline{D_3} \cdot \overline{D_2} + D_3 \cdot D_2)D_1 \cdot \overline{D_0} \\ = {} & (D_3 \oplus D_2)(\overline{D_1 \oplus D_0}) + (\overline{D_3 \oplus D_2})(D_1 \oplus D_0) \\ = {} & D_3 \oplus D_2 \oplus D_1 \oplus D_0 \end{aligned} \quad [3.57]$$

As the two outputs P_E and P_O are complementary, it follows that:

$$P_O = \overline{P_E} \quad [3.58]$$

Using XOR logic gates and an inverter, the logic circuit of a parity generator for 4-bit words is implemented as shown in Figure 3.54(a). Another version of the parity generator is given in Figure 3.54(b). The XOR output gate is configured as an inverter that can be programmed by the selection signal $\overline{E}/O$; this is useful to verify either the even parity, when $\overline{E}/O$ is set to 0, or the odd parity when $\overline{E}/O$ takes the logic level 1.

D_3	D_2	D_1	D_0	P_E	P_O
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

Table 3.28. *Truth table for a parity generator for 4-bit words*

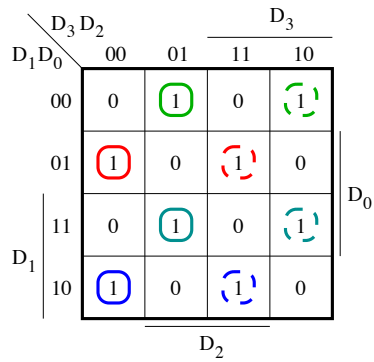


Figure 3.53. *Representation of* $P_E = D_3 \oplus D_2 \oplus D_1 \oplus D_0$

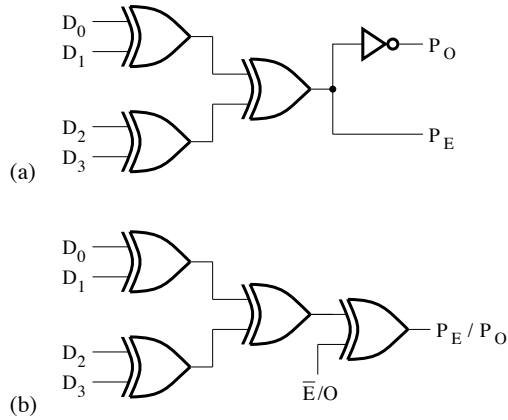


Figure 3.54. *Parity generator a) with or b) without selection signal*

The logic circuit shown in Figure 3.55 is a parity generator or checker that has select inputs to choose the desired parity type. Signal P takes logic level 1 if the input word contains an even number of bits set at 1. An analysis of the function table given in Table 3.29, where x denotes the don't care term, shows that each output signal (Σ_E or Σ_O) can be active high or low depending on the combination of logic levels at the select inputs.

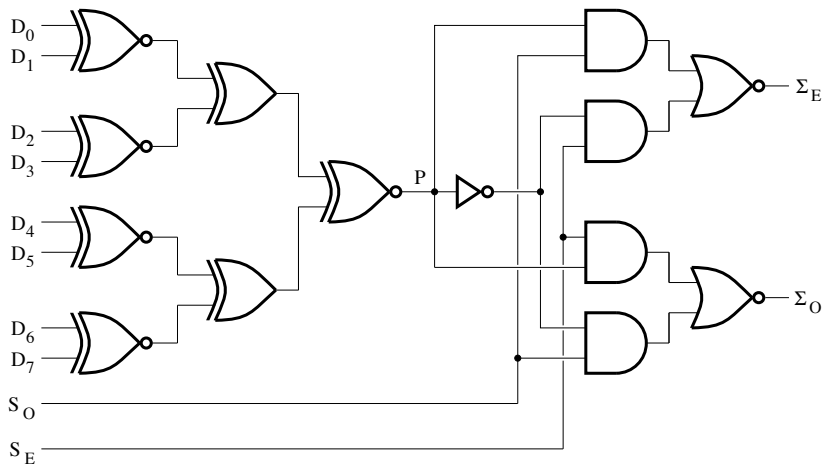


Figure 3.55. *Parity generator/checker SN74180*

Inputs			Outputs	
Number of bits, $D_0 - D_7$, set to 1	S_E	S_O	Σ_E	Σ_O
Even	1	0	1	0
Odd	1	0	0	1
Even	0	1	0	1
Odd	0	1	1	0
x	1	1	0	0
x	0	0	1	1

Table 3.29. *Table illustrating the operation of the parity generator/controller*

The configuration shown in Figure 3.56(a), where the input S_E is set to the logic level 1 and connected by an inverter to the input S_O, allows for the generation of an even parity bit. The 9-bit words are formed by concatenating the parity bit and the input word. The checker of even parity for 9-bit words is represented in Figure 3.56(b), where the parity bit and the complement of the parity bit are connected to inputs S_E and S_O, respectively. If an even parity exists, the output Σ_E takes the logic level 1 while the output Σ_O is set to 0.

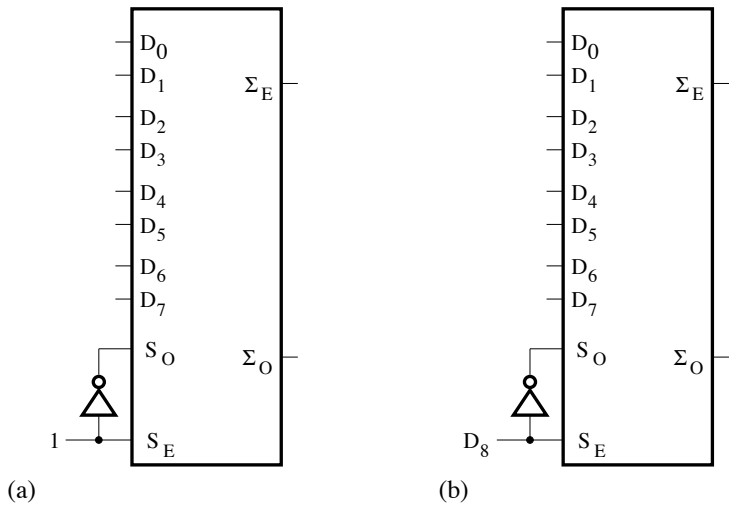


Figure 3.56. *a) Parity generator; b) parity checker for 9-bit words*

3.8. Barrel shifter

A barrel shifter is a logic circuit that is used to shift data bits by a certain number of positions to the left or right. There are logical shifts to the left or right, arithmetic shift to the right and rotation (or circular shift) to the right or left.

For logical shift, movement toward either end of a binary word is carried out by inserting 0-level bits at the other end, while for arithmetic shift, the sign bit or the MSB is duplicated and inserted each time at the left end. This translates to a loss of the bits originally at the ends of the binary words and whose number is equal to that of positions to be shifted.

Carrying out the following operations on a 4-bit code of the form $D_3D_2D_1D_0$, we obtain the following results:

– logic shift to the right by one position $0\,D_3D_2D_1$
– logic shift to the left by one position $D_2D_1D_0\,0$
– arithmetic shift to the right by one position $D_3D_3D_2D_1$
– rotation to the right by one position $D_0D_3D_2D_1$
– rotation to the left by one position $D_2D_1D_0D_3$

The barrel shifter is associated with the arithmetic and logic unit of some microprocessors to ensure a fast execution of shift and rotation operations.

A barrel shifter can be implemented by combining a certain number of multiplexer stages. Figure 3.57 shows the logic circuit of a barrel shift for 4-bit binary words. The input stage first reverses the order of data bits when $L = 1$. The intermediary stages then implement the shift or rotation to the right operation. Finally, the output stage carries out another inversion when $L = 1$ to yield the result. A 3-bit code (L, ROT and SRA) allows for the selection of shift and rotation operations as illustrated by Table 3.30, where *x* represents a don't care term.

Selection code			Operation
L	ROT	SRA	
0	0	0	Logic shift to the right
0	0	1	Logic shift to the right
0	1	x	Rotation to the right
1	0	0	Logic shift to the left
1	0	1	
1	1	x	Rotation to the left

Table 3.30. *Operations realized by the barrel shifter*

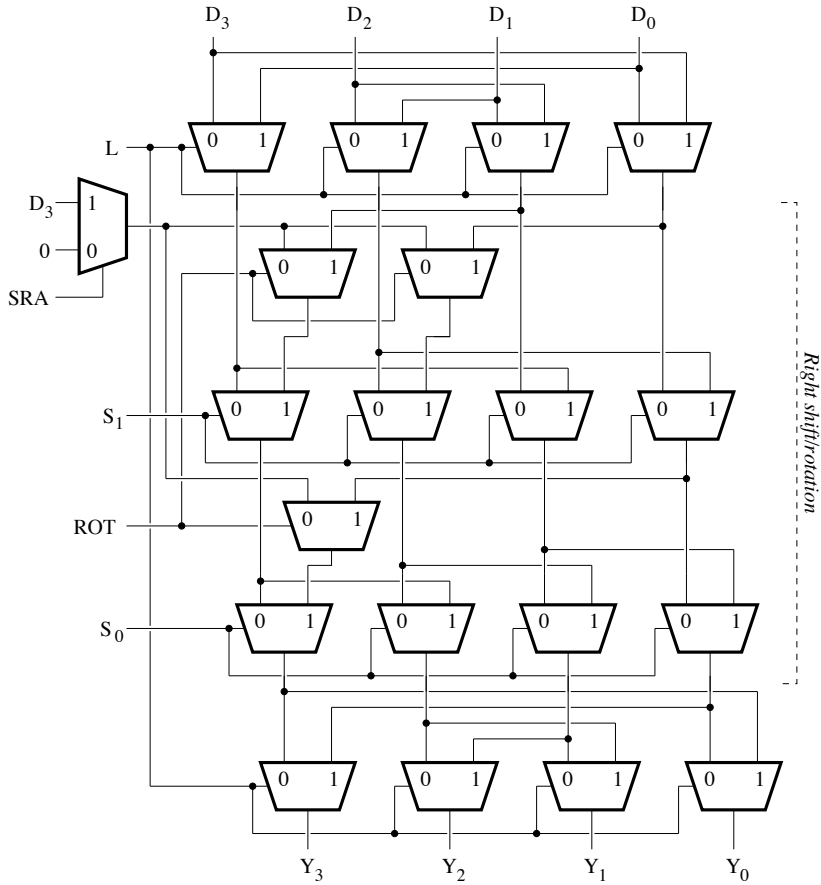


Figure 3.57. *Barrel shifter for 4-bit binary words*

The number of positions to be shifted is determined by a 2-bit code (S_1 and S_0). The logic equations for the outputs can be written as follows:

$$Y_0 = Q_0 \cdot \overline{L} + Q_3 \cdot L \qquad [3.59]$$

$$Y_1 = Q_1 \cdot \overline{L} + Q_2 \cdot L \qquad [3.60]$$

$$Y_2 = Q_2 \cdot \overline{L} + Q_1 \cdot L \qquad [3.61]$$

and

$$Y_3 = Q_3 \cdot \overline{L} + Q_0 \cdot L \qquad [3.62]$$

where:

$$Q_0 = (P_0 \cdot \overline{S_1} + P_2 \cdot S_1)\overline{S_0} + (P_1 \cdot \overline{S_1} + P_3 \cdot S_1)S_0 \quad [3.63]$$

$$Q_1 = (P_1 \cdot \overline{S_1} + P_3 \cdot S_1)S_0 + [P_2 \cdot \overline{S_1} + (D_3 \cdot SRA \cdot \overline{ROT} + P_0 \cdot ROT)S_1]S_0 \quad [3.64]$$

$$Q_2 = [P_2 \cdot \overline{S_1} + (D_3 \cdot SRA \cdot \overline{ROT} + P_0 \cdot ROT)S_1]\overline{S_0} + [P_3 \cdot \overline{S_1} + (D_3 \cdot SRA \cdot \overline{ROT} + P_1 \cdot ROT)S_1]S_0 \quad [3.65]$$

$$Q_3 = [P_3 \cdot \overline{S_1} + (D_3 \cdot SRA \cdot \overline{ROT} + P_1 \cdot ROT)S_1]\overline{S_0} + [D_3 \cdot SRA \cdot \overline{ROT} + (P_0 \cdot \overline{S_1} + P_2 \cdot S_1) \cdot ROT]S_0 \quad [3.66]$$

and we have:

$$P_0 = D_0 \cdot \overline{L} + D_3 \cdot L \quad [3.67]$$

$$P_1 = D_1 \cdot \overline{L} + D_2 \cdot L \quad [3.68]$$

$$P_2 = D_2 \cdot \overline{L} + D_1 \cdot L \quad [3.69]$$

$$P_3 = D_3 \cdot \overline{L} + D_0 \cdot L \quad [3.70]$$

Finally, we arrive at:

$$Y_0 = D_0 \cdot \overline{S_1} \cdot \overline{S_0} + (D_1 \cdot \overline{L} + D_3 \cdot L \cdot ROT)\overline{S_1} \cdot S_0 + [D_2(\overline{L} + ROT) + D_3 \cdot L \cdot SRA \cdot \overline{ROT}]S_1 \cdot \overline{S_0} + (D_1 \cdot L \cdot ROT + D_3 \cdot \overline{L})S_1 \cdot S_0 \quad [3.71]$$

$$Y_1 = D_0 \cdot \overline{S_1} \cdot \overline{S_0} + (D_2 \cdot \overline{L} + D_0 \cdot L)\overline{S_1} \cdot S_0 + D_3(\overline{L} + ROT + SRA)S_1 \cdot \overline{S_0} + [(D_0 \cdot \overline{L} + D_2 \cdot L)ROT + D_3 \cdot SRA \cdot \overline{ROT}]S_1 \cdot S_0 \quad [3.72]$$

$$Y_2 = D_2 \cdot \overline{S_1} \cdot \overline{S_0} + (D_3 \cdot \overline{L} + D_1 \cdot L)\overline{S_1} \cdot S_0 + [D_0(L + ROT) + D_3 \cdot \overline{L} \cdot SRA \cdot \overline{ROT}]S_1 \cdot \overline{S_0} + [(D_1 \cdot \overline{L} + D_3 \cdot L)ROT + D_3 \cdot SRA \cdot \overline{ROT}]S_1 \cdot S_0 \quad [3.73]$$

and

$$Y_3 = D_3 \cdot \overline{S_1} \cdot \overline{S_0} + [D_1(L + ROT) + D_3 \cdot \overline{L} \cdot SRA \cdot \overline{ROT}]S_1 \cdot \overline{S_0} + D_3 \cdot \overline{L} \cdot SRA \cdot \overline{ROT} \cdot S_0 + (D_0 \cdot \overline{L} \cdot ROT + D_2 \cdot L)\overline{S_1} \cdot S_0 + (D_2 \cdot \overline{L} \cdot ROT + D_0 \cdot L)S_1 \cdot S_0 \quad [3.74]$$

Considering the input word, $D_3D_2D_1D_0$, and L as entered variables, we obtain the truth table shown in Table 3.31.

Control inputs				Output bits			
ROT	SRA	S_1	S_0	Y_3	Y_2	Y_1	Y_0
x	x	0	0	D_3	D_2	D_1	D_0
0	0	0	1	$D_2 \cdot L$	$D_3 \cdot \overline{L} + D_1 \cdot L$	$D_2 \cdot \overline{L} + D_0 \cdot L$	$D_1 \cdot \overline{L}$
0	0	1	0	$D_1 \cdot L$	$D_0 \cdot L$	$D_3 \cdot \overline{L}$	$D_2 \cdot \overline{L}$
0	0	1	1	$D_0 \cdot L$	0	0	$D_3 \cdot \overline{L}$
0	1	0	1	$D_3 \cdot \overline{L} + D_2 \cdot L$	$D_3 \cdot \overline{L} + D_1 \cdot L$	$D_2 \cdot \overline{L} + D_0 \cdot L$	$D_1 \cdot \overline{L}$
0	1	1	0	$D_3 \cdot \overline{L} + D_1 \cdot L$	$D_3 \cdot \overline{L} + D_0 \cdot L$	D_3	$D_2 \cdot \overline{L} + D_3 \cdot L$
0	1	1	1	$D_3 \cdot \overline{L} + D_0 \cdot L$	D_3	D_3	$D_3 \cdot \overline{L}$
1	x	0	1	$D_0 \cdot \overline{L} + D_2 \cdot L$	$D_3 \cdot \overline{L} + D_1 \cdot L$	$D_2 \cdot \overline{L} + D_0 \cdot L$	$D_1 \cdot \overline{L} + D_3 \cdot L$
1	x	1	0	D_1	D_0	D_3	D_2
1	x	1	1	$D_2 \cdot \overline{L} + D_0 \cdot L$	$D_1 \cdot \overline{L} + D_3 \cdot L$	$D_0 \cdot \overline{L} + D_2 \cdot L$	$D_3 \cdot \overline{L} + D_1 \cdot L$

Table 3.31. *Truth table of the barrel shifter*

A barrel shifter can also be implemented using the $4:1$, $8:1$, or $16:1$ multiplexers. In general, increasing the size of the multiplexer makes it possible to add control signals, and hence more features.

EXAMPLE 3.2.– Four-bit barrel shifter

Implement a 4-bit barrel shifter that can shift or rotate the input data bits to the left. The barrel shifter is described by the following specifications:

– input data: $X_3\ X_2\ X_1\ X_0$;

– control data:

$R \in \{0, 1\}$: shift or rotate to the left;
$A_1\ A_0$: number of positions to be shifted;
$F \in \{0, 1\}$: bit to be inserted following shifting to the left.

– outputs: $Y_3\ Y_2\ Y_1\ Y_0$.

Table 3.32 depicts the truth table of the 4-bit barrel shifter. The logic function for each of the outputs Y_3, Y_2, Y_1 and Y_0 is characterized by each of the Karnaugh maps shown in Figure 3.58. The logic circuit of the 4-bit barrel shifter is represented in Figure 3.59, where an $8:1$ multiplexer is required to implement the logic function for each output.

R	A_1	A_0	Y_3	Y_2	Y_1	Y_0	
0	0	0	X_3	X_2	X_1	X_0	Transfer
0	0	1	X_2	X_1	X_0	F	sl1
0	1	0	X_1	X_0	F	F	sl2
0	1	1	X_0	F	F	F	sl3
1	0	0	X_3	X_2	X_1	X_0	Transfer
1	0	1	X_2	X_1	X_0	X_3	rl1
1	1	0	X_1	X_0	X_3	X_2	rl2
1	1	1	X_0	X_3	X_2	X_1	rl3

Table 3.32. *Truth table of the 4-bit barrel shifter*

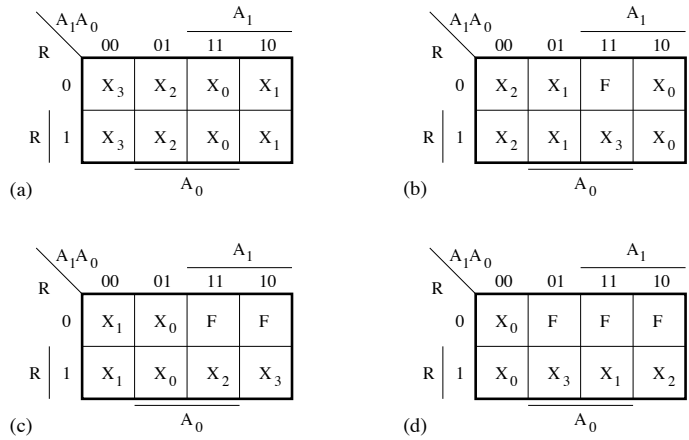


Figure 3.58. *Karnaugh maps: a) Y_3; b) Y_2; c) Y_1 and d) Y_0*

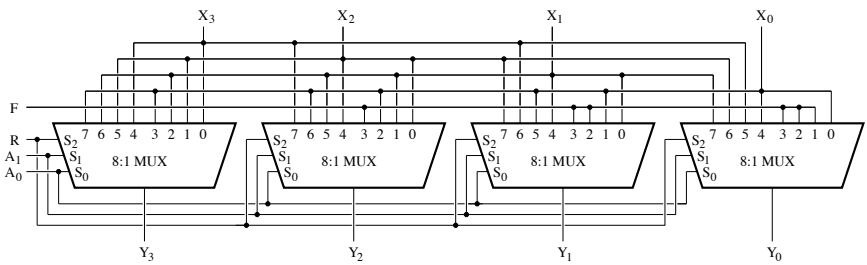


Figure 3.59. *Four-bit barrel shifter*

3.9. Exercises

EXERCISE 3.1.– Implementation of logic gates using $2:1$ multiplexer.

Show that the circuits in Figure 3.60(a)–3.60(d) are equivalent to the AND, OR, XOR and NAND logic gates, respectively.

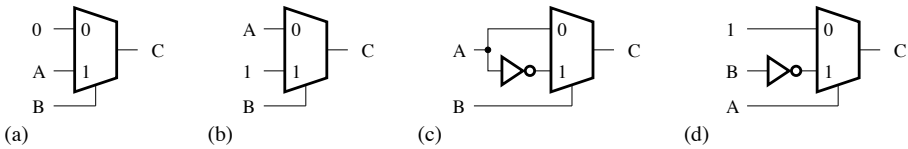


Figure 3.60. *Logic circuits with multiplexers*

EXERCISE 3.2.– Implementation of 2-out-of-4 decoder using 1-to-2 multiplexers.

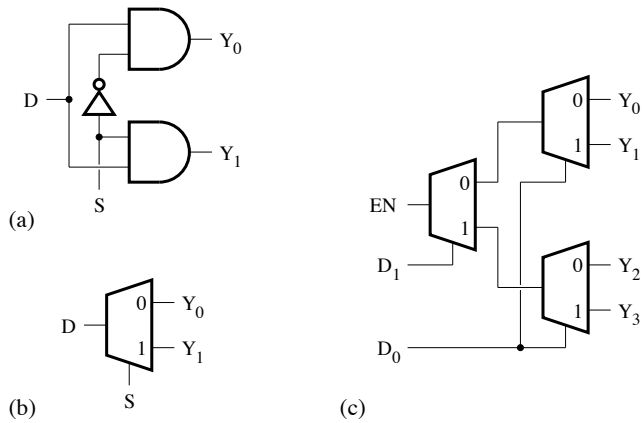


Figure 3.61. $2:4$ *decoder based on 1-to-2 multiplexers*

1) Verify that the logic circuit shown in Figure 3.61(a) implements a 1-to-2 demultiplexer, whose symbol is given in Figure 3.61(b).

2) To determine the function of the logic circuit illustrated in Figure 3.61(c), find the logic equations for the outputs and draw up the truth table.

EXERCISE 3.3.– Realization of 2-to-4 decoder using NAND gates.

To determine the role of the logic circuit shown in Figure 3.62, find the logic equations for the outputs and draw up the corresponding truth table.

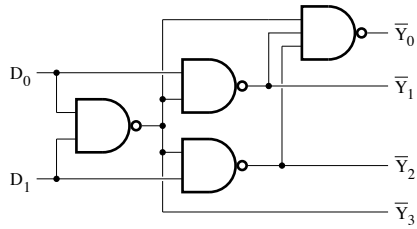


Figure 3.62. *2-out-of-4 encoder with NAND gates*

EXERCISE 3.4.– $8:3$ priority encoder.

To determine the role of the logic circuit shown in Figure 3.63, find the logic equations for the outputs and draw up the corresponding truth table.

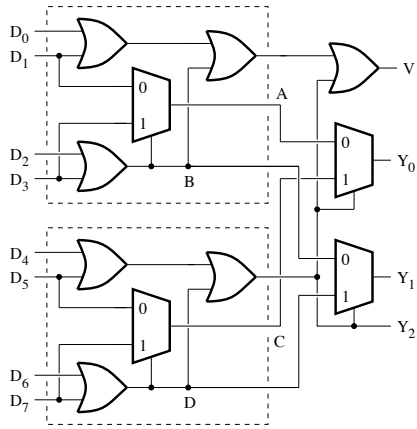


Figure 3.63. *8:3 priority encoder*

EXERCISE 3.5.– Implementation of the function $F(A, B, C)$.

Implement the logic function $F(A, B, C)$, which is characterized by the truth table shown in Table 3.33 using a $2:1$ multiplexer and logic gates to be determined.

EXERCISE 3.6.– 4-to-1 multiplexer.

Determine the logic equation for the output and construct the truth table for the multiplexer shown in Figure 3.64.

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.33. *Truth table of the function* $F(A, B, C)$

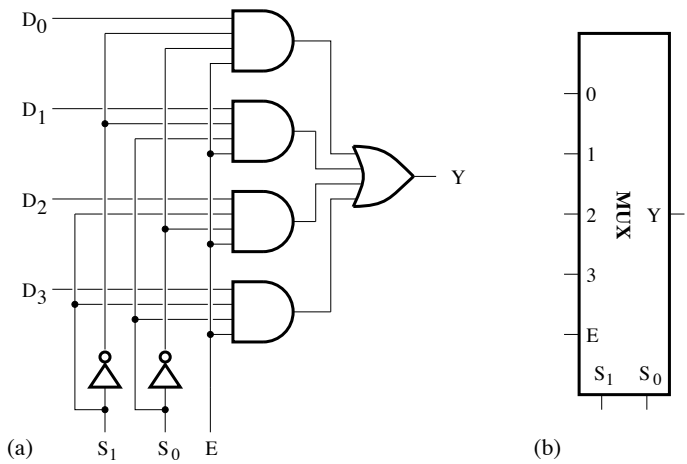


Figure 3.64. *a) 4-to-1 multiplexer with enable signal; b) symbol*

Use a 4-to-1 multiplexer and logic gates to implement the function:

$$F(A, B, C, D) = \sum m(3, 4, 5, 6, 7, 9, 10, 12, 14, 15) \qquad [3.75]$$

EXERCISE 3.7.– Demultiplexer/decoder.

Draw up the truth table for the decoder in Figure 3.65(a).

What is the difference between a decoder and demultiplexer?

Analyze the logic circuit shown in Figure 3.65(b) and determine the logic equations for the outputs $Z_i(A, B, C, D)$ with $i = 0, 1, 2, 3$.

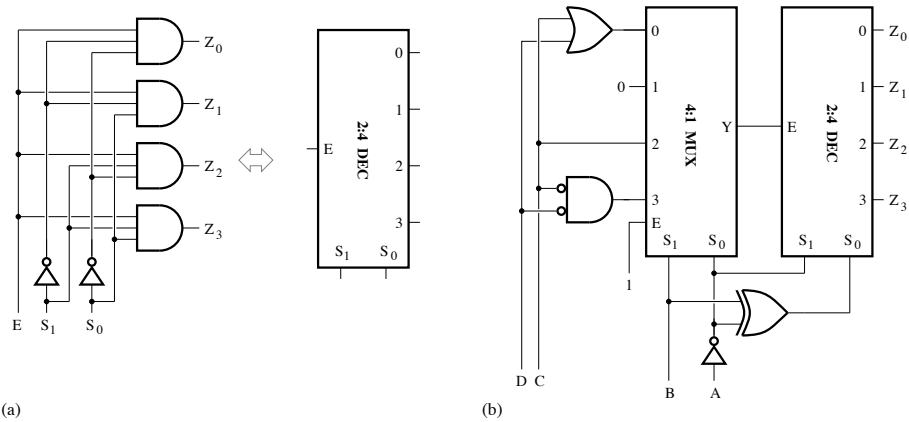


Figure 3.65. *a)* $2:4$ *decoder with an enable input; b) logic circuit*

EXERCISE 3.8.– Implementation of the function $F(A, B, C, D)$.

Implement the function, $F(A, B, C, D)$, whose logic expression is written as:

$$F(A, B, C, D) = A \oplus B \oplus C \oplus D \qquad [3.76]$$

using a $4:1$ multiplexer and logic gates to be determined.

EXERCISE 3.9.– Majority function of three variables.

The majority function, F, of three variables takes either the logic level 1, if the majority (two or three) of the variables is at logic level 1, or the logic level 0 in all other cases:

– construct the truth table of the function F;

– determine and simplify the logic equation for the function F;

– implement the function F using a $4:1$ multiplexer.

EXERCISE 3.10.– 8-to-1 multiplexer.

Verify by determining the logic equation for the output and by constructing the truth table when each of the logic circuits shown in Figure 3.66 works as an 8-to-1 multiplexer.

Use an 8-to-1 multiplexer and and logic gates to implement the following function:

$$F(A,B,C,D,E) = \sum m(0,1,2,4,5,6,7,13,14,20,21,22,28,29,30,31) \quad [3.77]$$

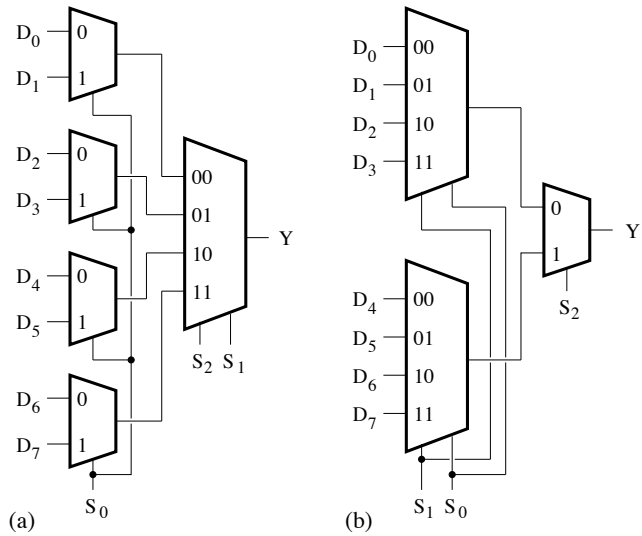


Figure 3.66. *Implementation of an 8-to-1 multiplexer based on a) four 2-to-1 multiplexers or b) two 4-to-1 multiplexers*

EXERCISE 3.11.– Implementation of a logic function based on a decoder

Implement the logic function:

$$F(A,B,C,D) = \sum m(1,3,7,9,15) \quad [3.78]$$

using a $3 : 8$ decoder with enable input $\overline{EN}$, a 5-input NAND gate and 2-input NAND gates. The decoder has active-low outputs.

EXERCISE 3.12.– Analysis of a logic circuit.

Determine the logic equation for each of the outputs Y_i $(i = 0, 1, 2, \cdots, 7)$ of the logic circuit shown in Figure 3.67.

Draw up the truth table for this circuit.

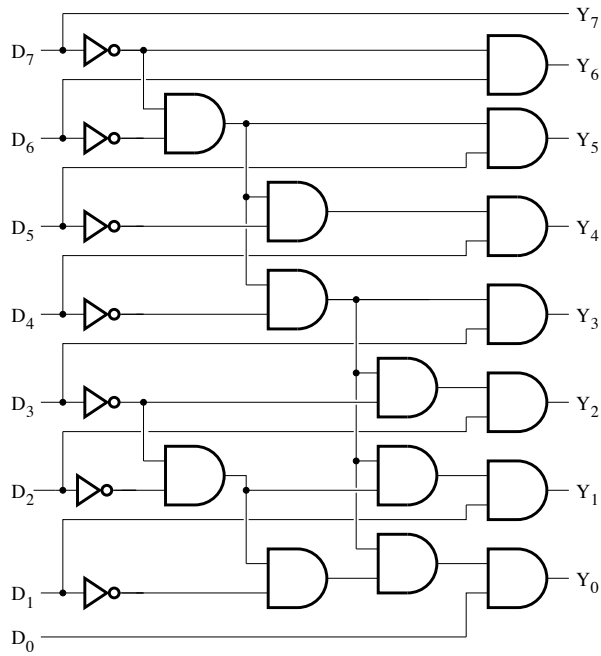


Figure 3.67. *Logic circuit*

What is the role of this circuit?

EXERCISE 3.13.– Design of a multiplier for 2-bit binary words.

We wish to implement a multiplier according to the following specifications:

– inputs: $X = X_1X_0$ and $Y = Y_1Y_0$;
– output: $Z = X \cdot Y$ where $Z = Z_3Z_2Z_1Z_0$.

Construct the truth table for this multiplier.

Determine the Boolean expressions for each of the outputs Z_i $(i = 0, 1, 2, 3)$.

Implement this multiplier using logic gates.

EXERCISE 3.14.– Comparator for 2-bit binary numbers.

Implement a comparator for 2-bit numbers: $P = AB$ and $Q = CD$ (see Figure 3.68).

Draw up the truth table for this comparator.

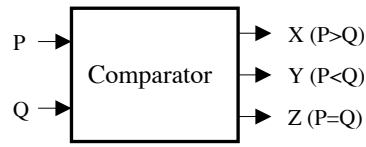


Figure 3.68. *Comparator*

Deduce and simplify the logic expressions for X, Y and Z.

Propose a logic circuit for the implementation of this comparator.

EXERCISE 3.15.– BCD-to-7-segment decoder.

A 4-bit number, A, B, C, D (D is the LSB), is applied to the inputs of the decoder supplying the signals a, b, c, d, e, f and g that are used to drive a 7-segment display (see Figure 3.69) generating numbers from 0 to 9.

Draw up the truth table for this decoder.

Deduce and simplify the logic expressions for a, b, c, d, e, f and g.

Propose a logic circuit that can be used to realize the decoder.

We will assume that the diodes of the display are controlled by low-level signals.

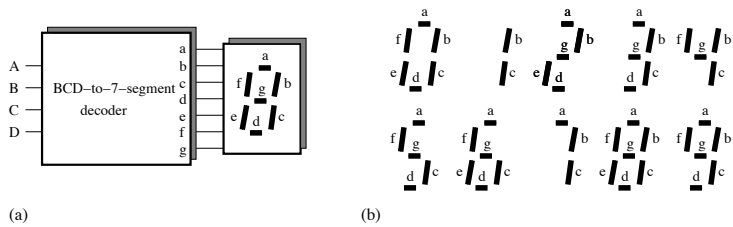


Figure 3.69. *a) BCD-to-7-segment decoder; b) display of numbers from 0 to 9*

EXERCISE 3.16.– HEX-to-7-segment decoder.

A number consisting of 4 bits, A, B, C and D (D is the LSB), is applied to the inputs of the decoder supplying the signals a, b, c, d, e, f and g that are used to drive a 7-segment display (see Figure 3.70) generating the numbers from 0 to 9 and the letters A–F corresponding to the hexadecimal representation of the numbers 10–15.

Draw up the truth table for the decoder, assuming that the diodes of the display are controlled by low-level signals.

Deduce and simplify the logic expressions for $\overline{a}$, $\overline{b}$, $\overline{c}$, $\overline{d}$, $\overline{e}$, $\overline{f}$ and $\overline{g}$.

Propose a logic circuit that can be used to implement the decoder, using logic gates with no more than four inputs.

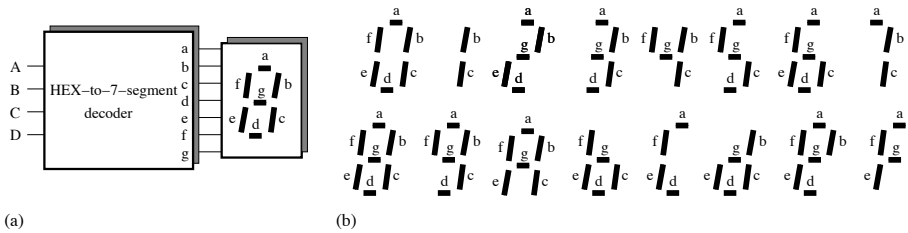


Figure 3.70. *a) HEX-to-7-segment decoder; b) display of numbers from 0 to 9 and letters A–F*

EXERCISE 3.17.– Analysis of logic circuits:

– analyze the logic circuit shown in Figure 3.71 and determine its function. The inputs are represented by X_4, X_3, X_2, X_1, X_0 and X_{-1}; the outputs by Y_3, Y_2, Y_1 and Y_0; and the control signals by D, S and E;

– same question for the logic circuit based on $2 : 1$ multiplexers and which is represented in Figure 3.72, where the data inputs are denoted by D_7, D_6, D_5, D_4, D_3, D_2, D_1 and D_0; the outputs by Y_7, Y_6, Y_5, Y_4, Y_3, Y_2, Y_1 and Y_0; and the control signals by S_2, S_1 and S_0.

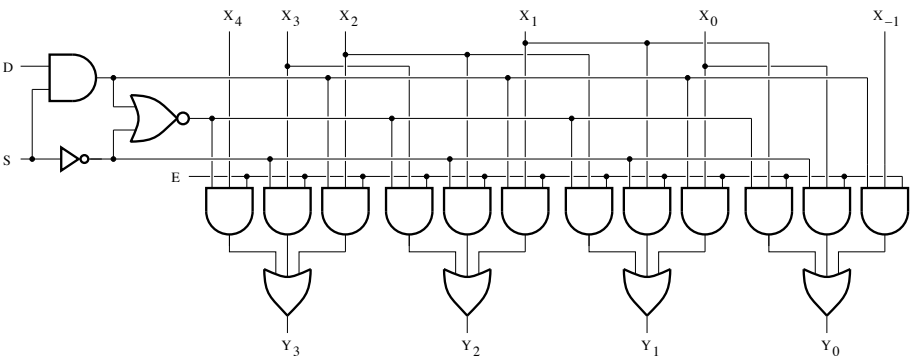


Figure 3.71. *Logic circuit 1*

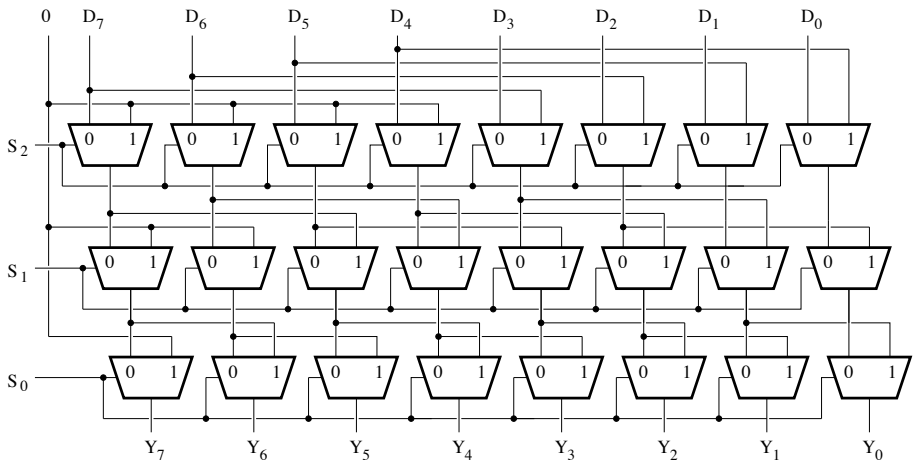


Figure 3.72. *Logic circuit 2*

3.10. Solutions

SOLUTION 3.1.– Implementation of a logic gate using $2:1$ multiplexer.

A $2:1$ multiplexer is characterized by a logic equation of the form:

$$Y = \overline{S} \cdot D_0 + S \cdot D_1 \qquad [3.79]$$

where the data input is represented by D_0 and D_1, and the selection input by S.

– AND gate: $D_0 = 0$, $D_1 = A$, $S = B$ and $Y = A \cdot B$;

– OR gate: $D_0 = A$, $D_1 = 1$, $S = B$ and $Y = A \cdot \overline{B} + B = A + B$;

– XOR gate: $D_0 = A$, $D_1 = \overline{A}$, $S = B$ and $Y = A \cdot \overline{B} + \overline{A} \cdot B = A \oplus B$;

– NAND gate: $D_0 = 1$, $D_1 = \overline{B}$, $S = A$ and $Y = \overline{A} + A \cdot \overline{B} = \overline{A} + \overline{B} = \overline{A \cdot B}$.

SOLUTION 3.2.– Implementation of a 2-out-of 4 decoder using 1-to-2 demultiplexers

1) 1-to-2 demultiplexers

The logic equations for the output are given by:

$$Y_0 = D \cdot \overline{S} \qquad \text{and} \qquad Y_1 = D \cdot S \qquad [3.80]$$

2) 2-out-of-4 Decoder

The logic equations fo the outputs are written as:

$$Y_0 = \overline{D_1} \cdot \overline{D_0} \cdot EN, \quad Y_1 = \overline{D_1} \cdot D_0 \cdot EN,$$
$$Y_2 = D_1 \cdot \overline{D_0} \cdot EN, \quad \text{and} \quad Y_3 = D_1 \cdot D_0 \cdot EN \qquad [3.81]$$

Table 3.34 gives the truth table of the 2-out-of-4 decoder.

S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	EN
0	1	0	0	EN	0
1	0	0	EN	0	0
1	1	EN	0	0	0

Table 3.34. *Truth table of the 2-out-of-4 decoder*

SOLUTION 3.3.– Realization of 2-out-of-4 decoder using NAND gates.

The logic equations for the outputs can be written as:

$$\overline{Y_0} = \overline{\overline{D_1 \cdot D_0} \cdot \overline{\overline{D_1} \cdot D_0} \cdot \overline{D_1 \cdot \overline{D_0}}}$$
$$= D_1 \cdot D_0 + \overline{D_1} \cdot D_0 + D_1 \cdot \overline{D_0} = D_1 + D_0 = \overline{\overline{D_1} \cdot \overline{D_0}} \qquad [3.82]$$
$$\overline{Y_1} = \overline{\overline{D_1 \cdot D_0} \cdot D_0} = D_1 \cdot D_0 + \overline{D_0} = D_1 + \overline{D_0} = \overline{\overline{D_1} \cdot D_0} \qquad [3.83]$$
$$\overline{Y_2} = \overline{D_1 \cdot \overline{D_1 \cdot D_0}} = \overline{D_1} + D_1 \cdot D_0 = \overline{D_1} + D_0 = \overline{D_1 \cdot \overline{D_0}} \qquad [3.84]$$
$$\text{and } \overline{Y_3} = \overline{D_1 \cdot D_0} \qquad [3.85]$$

The truth table of the 2-out-of-4 decoder, as represented in Table 3.35, corresponds to the case where outputs are active low.

D_1	D_0	$\overline{Y_3}$	$\overline{Y_2}$	$\overline{Y_1}$	$\overline{Y_0}$
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Table 3.35. *Truth table of the 2-out-of-2 decoder*

SOLUTION 3.4.– $8:3$ priority encoder.

The logic equations for the outputs are given by:

$$V = D_0 + D_1 + D_2 + D_3 + D_4 + D_5 + D_6 + D_7 \quad [3.86]$$

$$\begin{aligned} Y_0 &= A \cdot \overline{Y_2} + C \cdot Y_2 \\ &= [D_1(\overline{D_2 + D_3}) + D_3(D_2 + D_3)](\overline{D_4 + D_5 + D_6 + D_7}) \\ &\quad + [D_5(\overline{D_6 + D_7}) + D_7(D_6 + D_7)](D_4 + D_5 + D_6 + D_7) \\ &= (D_1 \cdot \overline{D_2} \cdot \overline{D_3} + D_3 \cdot D_2 + D_3)(\overline{D_4} \cdot \overline{D_5} \cdot \overline{D_6} \cdot \overline{D_7}) \\ &\quad + (D_5 \cdot \overline{D_6} \cdot \overline{D_7} + D_6 \cdot D_7 + D_7)(D_4 + D_5 + D_6 + D_7) \\ &= D_1 \cdot \overline{D_2} \cdot \overline{D_4} \cdot \overline{D_6} + D_3 \cdot \overline{D_4} \cdot \overline{D_6} + D_5 \cdot \overline{D_6} + D_7 \end{aligned} \quad [3.87]$$

$$\begin{aligned} Y_1 &= B \cdot \overline{Y_2} + D \cdot Y_2 \\ &= (D_2 + D_3)(\overline{D_4 + D_5 + D_6 + D_7}) \\ &\quad + (D_6 + D_7)(D_4 + D_5 + D_6 + D_7) \\ &= D_2 \cdot \overline{D_4} \cdot \overline{D_5} + D_3 \cdot \overline{D_4} \cdot \overline{D_5} + D_6 + D_7 \end{aligned} \quad [3.88]$$

and

$$Y_2 = D_4 + D_5 + D_6 + D_7 \quad [3.89]$$

Table 3.36 shows the truth table obtained from the logic equations of the $8:3$ priority encoder. The highest priority is assigned to the input corresponding to the highest decimal number. The validation output V is used to differentiate between the case where the input code corresponds to 0 and the case where no input is active.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_2	Y_1	Y_0	V
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	x	0	0	1	1
0	0	0	0	0	1	x	x	0	1	0	1
0	0	0	0	1	x	x	x	0	1	1	1
0	0	0	1	x	x	x	x	1	0	0	1
0	0	1	x	x	x	x	x	1	0	1	1
0	1	x	x	x	x	x	x	1	1	0	1
1	x	x	x	x	x	x	x	1	1	1	1
x	x	x	x	x	x	x	x	0	0	0	0

Table 3.36. *Truth table of the* $8:3$ *priority encoder*

SOLUTION 3.5.– Implementation of the function $F(A, B, C)$.

Analyzing the truth table shown in Table 3.37, we can deduce that the function F can take the following form:

$$F(A,B,C) = \begin{cases} \overline{B} & \text{if} \quad A = 0 \\ B + C & \text{if} \quad A = 1 \end{cases} \qquad [3.90]$$

A	B	C	F	
0	0	0	1	$\overline{B}$
0	0	1	1	
0	1	0	0	
0	1	1	0	
1	0	0	0	$B + C$
1	0	1	1	
1	1	0	1	
1	1	1	1	

Table 3.37. *Truth table of the logic function F*

Similarly, using Shannon's theorem, we can write:

$$F(A,B,C) = \overline{A} \cdot F(0,B,C) + A \cdot F(1,B,C) \qquad [3.91]$$

where $F(0,B,C) = \overline{B}$ and $F(1,B,C) = B + C$.

The logic function F can, thus, be implemented as shown in Figure 3.73 using a $2:1$ multiplexer, an OR gate and an inverter.

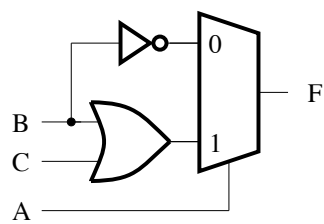
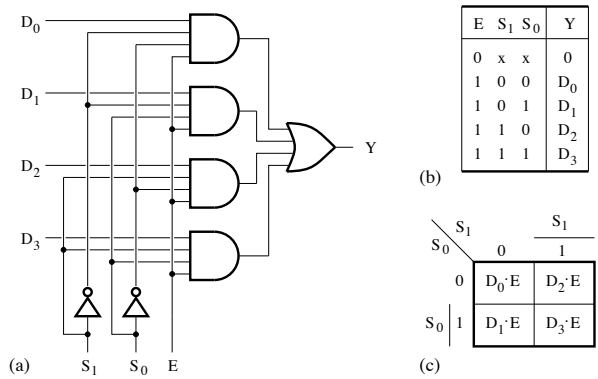


Figure 3.73. *Implementation of the logic function F*

SOLUTION 3.6.– 4-to-1 multiplexer.

The logic equation for the output of the multiplexer with enable signal is given by:

$$Y = \overline{S_1} \cdot \overline{S_0} \cdot D_0 \cdot E + \overline{S_1} \cdot S_0 \cdot D_1 \cdot E + S_1 \cdot \overline{S_0} \cdot D_2 \cdot E + S_1 \cdot S_0 \cdot D_3 \cdot E \quad [3.92]$$



E	S_1	S_0	Y
0	x	x	0
1	0	0	D_0
1	0	1	D_1
1	1	0	D_2
1	1	1	D_3

S_0 \ S_1	0	1
0	$D_0 \cdot E$	$D_2 \cdot E$
1	$D_1 \cdot E$	$D_3 \cdot E$

Figure 3.74. *4-to-1 multiplexer: a) logic circuit, b) truth table and c) Karnaugh maps for the output Y*

Implementation of the function $F(A, B, C, D)$

The logic function $F(A, B, C, D)$ can be written as follows:

$$F(A,B,C,D) = \sum m(3,4,5,6,7,9,10,12,14,15) \quad [3.93]$$

$$= \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot C \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot D \quad [3.94]$$

$$= \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot B(\overline{C} \cdot \overline{D} + \overline{C} \cdot D + C \cdot \overline{D} + C \cdot D) + A \cdot \overline{B}(\overline{C} \cdot D + C \cdot \overline{D}) + A \cdot B(\overline{C} \cdot \overline{D} + C \cdot \overline{D} + C \cdot D) \quad [3.95]$$

$$= \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot B + A \cdot \overline{B}(C \oplus D) + A \cdot B(C + \overline{D}) \quad [3.96]$$

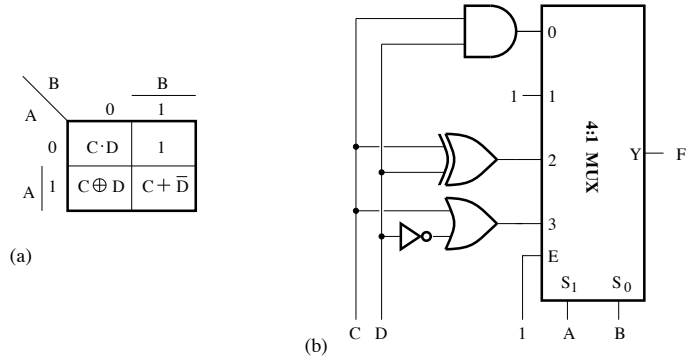


Figure 3.75. *a) Karnaugh maps for F; b) logic circuit implementing F*

SOLUTION 3.7.– Demultiplexer/decoder.

The following aspects make it possible to differentiate between a decoder and a demultiplexer:

– n-out-of-2^n decoder: n input data and 2^n outputs;

– 1-to-2^n demultiplexer: 1 data input, n select inputs and 2^n outputs.

A decoder with enable input E operates like a demultiplexer if E is used as a data input.

The logic equations for the outputs of the 2-out-of-4 decoders shown in Figure 3.76 are given by:

$$Z_0 = \overline{S_1} \cdot \overline{S_0} \cdot E \quad [3.97]$$

$$Z_1 = \overline{S_1} \cdot S_0 \cdot E \quad [3.98]$$

$$Z_2 = S_1 \cdot \overline{S_0} \cdot E \quad [3.99]$$

$$Z_3 = S_1 \cdot S_0 \cdot E \quad [3.100]$$

Table 3.38 gives the truth table of the 2-out-of-4 decoder.

Applying the data sequence D to input E, we can implement a 1-to-4 demultiplexer as shown in Figure 3.77. Table 3.39 gives the truth table of the 1-to-4 demultiplexer.

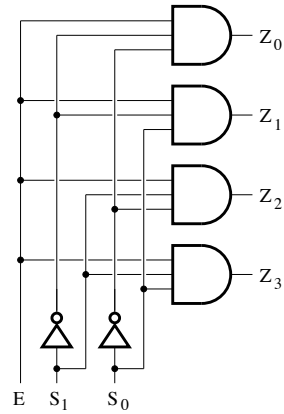


Figure 3.76. *2-out-of-4 decoder*

E	S_1	S_0	Z_3	Z_2	Z_1	Z_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Table 3.38. *Truth table of the 2-out-of-4 decoder*

Analyzing the logic circuit made up of a 2-out-of-4 decoder, a 1-to-4 multiplexer and logic gates makes it possible to obtain the logic equation of the multiplexer output:

$$Y = [\overline{S_1} \cdot \overline{S_0}(C + D) + \overline{S_1} \cdot S_0(0) + S_1 \cdot \overline{S_0} \cdot C + S_1 \cdot S_0 \cdot \overline{C} \cdot \overline{D}\,]E \quad [3.101]$$

where $E = 1$, $S_0 = \overline{A}$ and $S_1 = B$, and the logic equations of the decoder outputs:

$$Z_0 = \overline{S_1} \cdot \overline{S_0} \cdot E \quad [3.102]$$

$$Z_1 = \overline{S_1} \cdot S_0 \cdot E \quad [3.103]$$

$$Z_2 = S_1 \cdot \overline{S_0} \cdot E \quad [3.104]$$

$$Z_3 = S_1 \cdot S_0 \cdot E \quad [3.105]$$

where $E = Y$, $S_0 = \overline{A} \oplus B$ and $S_1 = \overline{A}$. Combining the above-mentioned equations, we arrive at:

$$Z_0 = A \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot D \quad [3.106]$$

$$Z_1 = A \cdot B \cdot C \quad [3.107]$$

$$Z_2 = \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} \quad [3.108]$$

$$Z_3 = 0 \quad [3.109]$$

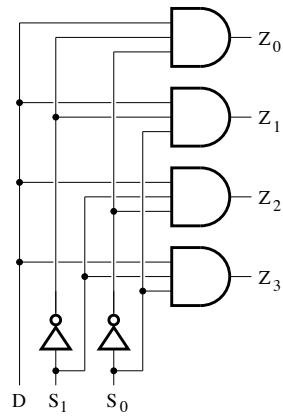


Figure 3.77. *1-to-4 demultiplexer*

S_1	S_0	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Table 3.39. *Truth table of the 1-to-4 demultiplexer*

SOLUTION 3.8.– Implementation of the function $F(A, B, C, D)$.

Using Shannon's theorem, the logic function F may be decomposed as follows:

$$F(A, B, C, D) = A \oplus B \oplus C \oplus D \quad [3.110]$$

$$= (A \oplus B \oplus 1 \oplus 1)C \cdot D + (A \oplus B \oplus 1 \oplus 0)C \cdot \overline{D} + (A \oplus B \oplus 0 \oplus 1)\overline{C} \cdot D + (A \oplus B \oplus 0 \oplus 0)\overline{C} \cdot \overline{D} \quad [3.111]$$

$$= (A \oplus B)C \cdot D + (\overline{A \oplus B})C \cdot \overline{D} +$$
$$(\overline{A \oplus B})\overline{C} \cdot D + (A \oplus B)\overline{C} \cdot \overline{D} \quad [3.112]$$

Figure 3.78 shows the logic circuit for the implementation of the function F using a $4:1$ multiplexer, an XOR gate and an inverter.

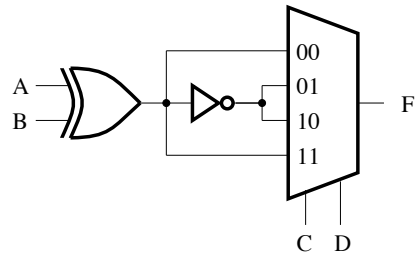


Figure 3.78. *Logic circuit for the function F*

SOLUTION 3.9.– Majority function of three variables.

The truth table for the majority function of three variables is represented in Table 3.40. Hence, we have:

$$F = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C \quad [3.113]$$
$$= (\overline{A} + A)B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$
$$= (B + \overline{B} \cdot A)C + A \cdot B \cdot \overline{C}$$
$$= B \cdot C + A(C + \overline{C} \cdot B)$$
$$= B \cdot C + A \cdot C + A \cdot B \quad [3.114]$$

The truth table for the majority function of three variables is represented in Table 3.40. Thus:

$$F = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C \quad [3.115]$$
$$= (\overline{A} + A)B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$
$$= (B + \overline{B} \cdot A)C + A \cdot B \cdot \overline{C}$$
$$= B \cdot C + A(C + \overline{C} \cdot B)$$
$$= B \cdot C + A \cdot C + A \cdot B \quad [3.116]$$

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.40. *Truth table*

Inserting different combinations of the variables A and B in the logic equation for the function F makes it possible to construct the reduced truth table shown in Table 3.41.

A	B	F
0	0	0
0	1	C
1	0	C
1	1	1

Table 3.41. *Truth table*

Figure 3.79 depicts the logic circuit that can be used to implement the majority function of three variables.

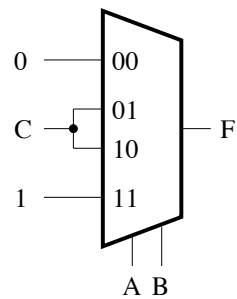


Figure 3.79. *Logic circuit*

SOLUTION 3.10.– 8-to-1 multiplexer

For the first circuit, we have:

$$\begin{aligned} Y &= \overline{S_2} \cdot \overline{S_1}(D_0 \cdot \overline{S_0} + D_1 \cdot S_0) + \overline{S_2} \cdot S_1(D_2 \cdot \overline{S_0} + D_3 \cdot S_0) \\ &\quad + S_2 \cdot \overline{S_1}(D_4 \cdot \overline{S_0} + D_5 \cdot S_0) + S_2 \cdot S_1(D_6 \cdot \overline{S_0} + D_7 \cdot S_0) \end{aligned} \quad [3.117]$$

and for the second, we have:

$$\begin{aligned} Y &= \overline{S_2}(\overline{S_1} \cdot \overline{S_0} \cdot D_0 + \overline{S_1} \cdot S_0 \cdot D_1 + S_1 \cdot \overline{S_0} \cdot D_2 + S_1 \cdot S_0 \cdot D_3) \\ &\quad + S_2(\overline{S_1} \cdot \overline{S_0} \cdot D_4 + \overline{S_1} \cdot S_0 \cdot D_5 + S_1 \cdot \overline{S_0} \cdot D_6 + S_1 \cdot S_0 \cdot D_7) \end{aligned} \quad [3.118]$$

In both cases, the logic equation for the output can be put into the following form:

$$\begin{aligned} Y &= \overline{S_2} \cdot \overline{S_1} \cdot \overline{S_0} \cdot D_0 + \overline{S_2} \cdot \overline{S_1} \cdot S_0 \cdot D_1 + \\ &\quad \overline{S_2} \cdot S_1 \cdot \overline{S_0} \cdot D_2 + \overline{S_2} \cdot S_1 \cdot S_0 \cdot D_3 + S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot D_4 + \\ &\quad S_2 \cdot \overline{S_1} \cdot S_0 \cdot D_5 + S_2 \cdot S_1 \cdot \overline{S_0} \cdot D_6 + S_2 \cdot S_1 \cdot S_0 \cdot D_7 \end{aligned} \quad [3.119]$$

Table 3.42 gives the truth table of the 8-to-1 multiplexer.

S_0	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Table 3.42. *Truth table of the 8-to-1 multiplexer*

$$F(A,B,C,D,E) = \sum m(0,1,2,4,5,6,7,13,14,20,21,22,28,29,30,31) \quad [3.120]$$

$$\begin{aligned} &= \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot \overline{E} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot E + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D \cdot \overline{E} + \\ &\quad \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} \cdot \overline{E} + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} \cdot E + \overline{A} \cdot \overline{B} \cdot C \cdot D \cdot \overline{E} + \\ &\quad \overline{A} \cdot \overline{B} \cdot C \cdot D \cdot E + \overline{A} \cdot B \cdot C \cdot \overline{D} \cdot E + \overline{A} \cdot B \cdot C \cdot D \cdot \overline{E} + \end{aligned}$$

$$A \cdot \overline{B} \cdot C \cdot \overline{D} \cdot \overline{E} + A \cdot \overline{B} \cdot C \cdot \overline{D} \cdot E + A \cdot \overline{B} \cdot C \cdot D \cdot \overline{E} +$$
$$A \cdot B \cdot C \cdot \overline{D} \cdot \overline{E} + A \cdot B \cdot C \cdot \overline{D} \cdot E + A \cdot B \cdot C \cdot D \cdot \overline{E} +$$
$$A \cdot B \cdot C \cdot D \cdot E \qquad [3.121]$$
$$= (\overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C)(\overline{D} \cdot \overline{E} + \overline{D} \cdot E + D \cdot \overline{E}) +$$
$$(\overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot C)(\overline{D} \cdot \overline{E} + \overline{D} \cdot E + D \cdot \overline{E} + D \cdot E) +$$
$$\overline{A} \cdot B \cdot C(\overline{D} \cdot E + D \cdot \overline{E}) \qquad [3.122]$$

and:

$$F(A, B, C, D, E) = (\overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C)(\overline{D \cdot E}) +$$
$$(\overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot C)(1) + \overline{A} \cdot B \cdot C(D \oplus E) \qquad [3.123]$$

Choosing A, B and C, as selection inputs, we can obtain the Karnaugh map shown in Figure 3.80(a) that can be used to determine the combination of variables to be applied to the data inputs of the multiplexer. Figure 3.80(b) depicts the logic circuit that can be used to implement the function F.

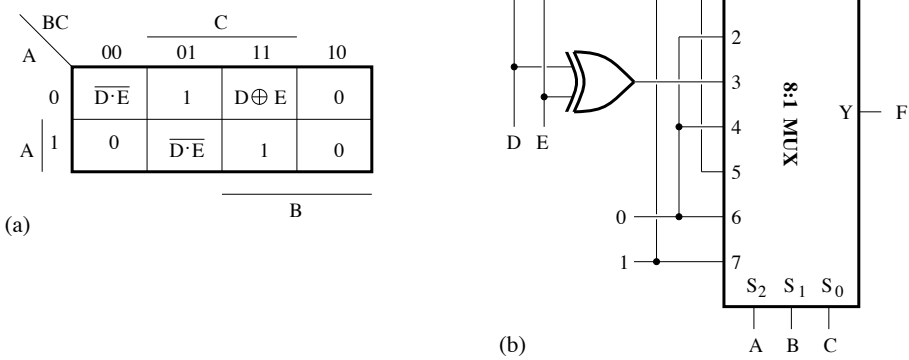


Figure 3.80. *Implementation of F: a) Karnaugh map; b) logic circuit*

SOLUTION 3.11.– Implementation of a logic function using a decoder.

The logic function F can be expressed in the following form:

$$F(A,B,C,D) = \sum m(1,3,7,9,15) \quad [3.124]$$

$$= \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D + \overline{A}\cdot\overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot C\cdot D + A\cdot\overline{B}\cdot\overline{C}\cdot D + A\cdot B\cdot C\cdot D \quad [3.125]$$

To implement the function F using NAND logic gates and a $3:8$ decoder whose logic equations can be written as follows:

$$\begin{aligned} Y_0 &= \overline{X_2}\cdot\overline{X_1}\cdot\overline{X_0}\cdot\overline{EN} \quad Y_1 = \overline{X_2}\cdot\overline{X_1}\cdot X_0\cdot\overline{EN} \quad Y_2 = \overline{X_2}\cdot X_1\cdot\overline{X_0}\cdot\overline{EN} \\ Y_3 &= \overline{X_2}\cdot X_1\cdot X_0\cdot\overline{EN} \quad Y_4 = X_2\cdot\overline{X_1}\cdot\overline{X_0}\cdot\overline{EN} \quad Y_5 = X_2\cdot\overline{X_1}\cdot X_0\cdot\overline{EN} \\ Y_6 &= X_2\cdot X_1\cdot\overline{X_0}\cdot\overline{EN} \quad Y_7 = X_2\cdot X_1\cdot X_0\cdot\overline{EN} \end{aligned} \quad [3.126]$$

we observe that:

$$F = Y_0 + Y_1 + Y_3 + Y_4 + Y_7 \quad [3.127]$$

$$= \overline{\overline{Y_0}\cdot\overline{Y_1}\cdot\overline{Y_3}\cdot\overline{Y_4}\cdot\overline{Y_7}} \quad [3.128]$$

where:

$$\begin{aligned} Y_0 &= \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D \quad Y_1 = \overline{A}\cdot\overline{B}\cdot C\cdot D \quad Y_3 = \overline{A}\cdot B\cdot C\cdot D \\ Y_4 &= A\cdot\overline{B}\cdot\overline{C}\cdot D \quad Y_7 = A\cdot B\cdot C\cdot D \end{aligned} \quad [3.129]$$

Figure 3.81 gives the logic circuit that can be used to realize the function F.

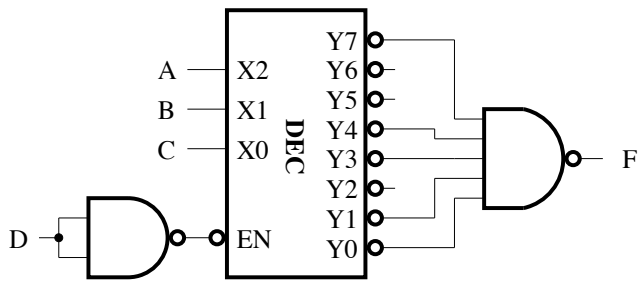


Figure 3.81. *Implementation of the logic function F*

SOLUTION 3.12.– Analysis of a logic circuit.

The output equations obtained by analyzing the proposed logic circuit are given by:

$$Y_7 = D_7 \quad [3.130]$$

$$Y_6 = \overline{D_7} \cdot D_6 \quad [3.131]$$

$$Y_5 = \overline{D_7} \cdot \overline{D_6} \cdot D_5 \quad [3.132]$$

$$Y_4 = \overline{D_7} \cdot \overline{D_6} \cdot \overline{D_5} \cdot D_4 \quad [3.133]$$

$$Y_3 = \overline{D_7} \cdot \overline{D_6} \cdot \overline{D_5} \cdot \overline{D_4} \cdot D_3 \quad [3.134]$$

$$Y_2 = \overline{D_7} \cdot \overline{D_6} \cdot \overline{D_5} \cdot \overline{D_4} \cdot \overline{D_3} \cdot D_2 \quad [3.135]$$

$$Y_1 = \overline{D_7} \cdot \overline{D_6} \cdot \overline{D_5} \cdot \overline{D_4} \cdot \overline{D_3} \cdot \overline{D_2} \cdot D_1 \quad [3.136]$$

and:

$$Y_0 = \overline{D_7} \cdot \overline{D_6} \cdot \overline{D_5} \cdot \overline{D_4} \cdot \overline{D_3} \cdot \overline{D_2} \cdot \overline{D_1} \cdot D_0 \quad [3.137]$$

The truth table based on the logic equations for the outputs is given in Table 3.43.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	x	0	0	0	0	0	0	1	0
0	0	0	0	0	1	x	x	0	0	0	0	0	1	0	0
0	0	0	0	1	x	x	x	0	0	0	0	1	0	0	0
0	0	0	1	x	x	x	x	0	0	0	1	0	0	0	0
0	0	1	x	x	x	x	x	0	0	1	0	0	0	0	0
0	1	x	x	x	x	x	x	0	1	0	0	0	0	0	0
1	x	x	x	x	x	x	x	1	0	0	0	0	0	0	0

Table 3.43. *Truth table*

This is a priority selector that can be connected to an $8:3$ encoder in order to realize an $8:3$ priority encoder.

SOLUTION 3.13.– Design of a multiplier for 2-bit words.

Referring to the truth table shown in Table 3.44, we can obtain the following logic equations:

$$\begin{aligned} Z_0 &= \overline{X_1} \cdot X_0 \cdot \overline{Y_1} \cdot Y_0 + \overline{X_1} \cdot X_0 \cdot Y_1 \cdot Y_0 + X_1 \cdot X_0 \cdot \overline{Y_1} \cdot Y_0 \\ &\quad + X_1 \cdot X_0 \cdot Y_1 \cdot Y_0 \\ &= X_0 \cdot Y_0 \end{aligned} \quad [3.138]$$

$$\begin{aligned} Z_1 &= \overline{X_1} \cdot X_0 \cdot Y_1 \cdot \overline{Y_0} + \overline{X_1} \cdot X_0 \cdot Y_1 \cdot Y_0 + X_1 \cdot \overline{X_0} \cdot \overline{Y_1} \cdot Y_0 \\ &\quad + X_1 \cdot \overline{X_0} \cdot Y_1 \cdot Y_0 + X_1 \cdot X_0 \cdot \overline{Y_1} \cdot Y_0 + X_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y_0} \\ &= \overline{X_1} \cdot X_0 \cdot Y_1 + X_0 \cdot Y_1 \cdot \overline{Y_0} + X_1 \cdot \overline{X_0} \cdot Y_0 + X_1 \cdot \overline{Y_1} \cdot Y_0 \end{aligned} \quad [3.139]$$

$$\begin{aligned} Z_2 &= X_1 \cdot \overline{X_0} \cdot Y_1 \cdot \overline{Y_0} + X_1 \cdot \overline{X_0} \cdot Y_1 \cdot Y_0 + X_1 \cdot X_0 \cdot Y_1 \cdot \overline{Y_0} \\ &= X_1 \cdot \overline{X_0} \cdot Y_1 + X_1 \cdot Y_1 \cdot Y_0 \end{aligned} \quad [3.140]$$

and

$$Z_3 = X_1 \cdot X_0 \cdot Y_1 \cdot Y_0 \quad [3.141]$$

$X \cdot Y = Z$	X		Y		Z			
	X_1	X_0	Y_1	Y_0	Z_3	Z_2	Z_1	Z_0
$0 \cdot 0 = 0$	0	0	0	0	0	0	0	0
$0 \cdot 1 = 0$	0	0	0	1	0	0	0	0
$0 \cdot 2 = 0$	0	0	1	0	0	0	0	0
$0 \cdot 3 = 0$	0	0	1	1	0	0	0	0
$1 \cdot 0 = 0$	0	1	0	0	0	0	0	0
$1 \cdot 1 = 1$	0	1	0	1	0	0	0	1
$1 \cdot 2 = 2$	0	1	1	0	0	0	1	0
$1 \cdot 3 = 3$	0	1	1	1	0	0	1	1
$2 \cdot 0 = 0$	1	0	0	0	0	0	0	0
$2 \cdot 1 = 2$	1	0	0	1	0	0	1	0
$2 \cdot 2 = 4$	1	0	1	0	0	1	0	0
$2 \cdot 3 = 6$	1	0	1	1	0	1	1	0
$3 \cdot 0 = 0$	1	1	0	0	0	0	0	0
$3 \cdot 1 = 3$	1	1	0	1	0	0	1	1
$3 \cdot 2 = 6$	1	1	1	0	0	1	1	0
$3 \cdot 3 = 9$	1	1	1	1	1	0	0	1

Table 3.44. *Truth table of a multiplier for 2-bit words*

These equations can then be used to construct the logic circuit of a multiplier for 2-bit words as shown in Figure 3.82.

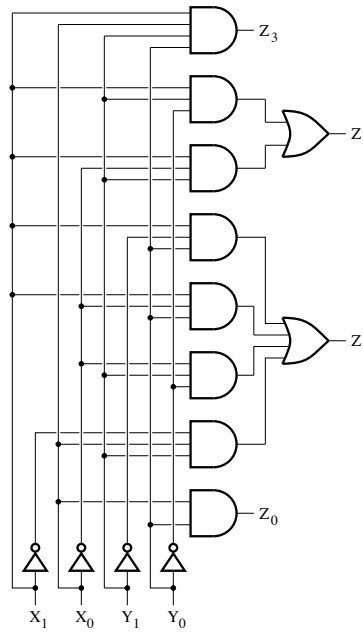


Figure 3.82. *Multiplier for 2-bit words*

SOLUTION 3.14.– Comparator for 2-bit numbers.

For the outputs of the comparator, the logic equations obtained from the truth table shown in Table 3.45 are given by:

$$\begin{aligned} X &= \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot D \\ &\quad + A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot C \cdot \overline{D} \\ &= \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{C}(\overline{B} \cdot \overline{D} + \overline{B} \cdot D + B \cdot \overline{D} + B \cdot D) + A \cdot B \cdot C \cdot \overline{D} \\ &= A \cdot \overline{C} + A \cdot B \cdot \overline{D} + B \cdot \overline{C} \cdot \overline{D} \end{aligned} \quad [3.142]$$

$$\begin{aligned} Y &= \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot D \\ &\quad + \overline{A} \cdot B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot C \cdot D \\ &= \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot C(\overline{B} \cdot \overline{D} + \overline{B} \cdot D + B \cdot \overline{D} + B \cdot D) + A \cdot \overline{B} \cdot C \cdot D \\ &= \overline{A} \cdot C + \overline{A} \cdot \overline{B} \cdot D + \overline{B} \cdot C \cdot D \end{aligned} \quad [3.143]$$

and

$$Z = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot C \cdot D + A \cdot \overline{B} \cdot C \cdot \overline{D} \quad [3.144]$$

A	B	C	D	X	Y	Z
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	1

Table 3.45. *Truth table of a comparator for 2-bit numbers*

It should be noted that:

$$\overline{B} \cdot \overline{D} + \overline{B} \cdot D + B \cdot \overline{D} + B \cdot D = 1$$

Figure 3.83 depicts the logic circuit of the comparator for 2-bit numbers.

SOLUTION 3.15.– BCD-to-7-segment decoder.

Table 3.46 presents the truth table of the BCD-to-7-segment decoder that can be used to determine the output logic equations.

Thus, for each of the outputs we have:

$$a = \sum m(0, 2, 3, 5, 6, 7, 8, 9) \quad [3.145]$$

$$b = \sum m(0, 1, 2, 3, 4, 7, 8, 9) \quad [3.146]$$

$$c = \sum m(0, 1, 3, 4, 5, 6, 7, 8, 9) \quad [3.147]$$

$$d = \sum m(0, 2, 3, 5, 6, 8, 9) \quad [3.148]$$

$$e = \sum m(0, 2, 6, 8) \quad [3.149]$$

$$f = \sum m(0, 4, 5, 6, 8, 9) \quad [3.150]$$

and

$$g = \sum m(2, 3, 4, 5, 6, 8, 9) \quad [3.151]$$

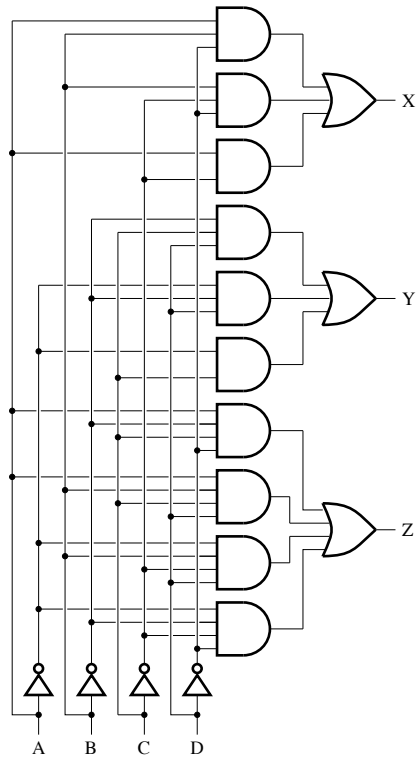


Figure 3.83. *Logic circuit of a comparator for 2-bit numbers*

Because the LEDs are driven by low-level signals, the minimal form in the sum of products is obtained by constructing the corresponding Karnaugh maps in the complemented forms for each of output variables (see Figures 3.84–3.90).

The logic circuit for the BCD-to-7-segment decoder is represented in Figure 3.91.

SOLUTION 3.16.– HEX-to-7-segment decoder.

The number of hexadecimal digits to be represented, that is 16, is equal to number of possible combinations with four bits. The truth table of the HEX-to-7-segment decoder can be constructed as shown in Table 3.47. We can then obtain the following logic equations:

$$\overline{a} = \sum m(1, 4, 11, 13) \qquad [3.152]$$

$$\overline{b} = \sum m(5, 6, 11, 12, 15) \qquad [3.153]$$

$$\overline{c} = \sum m(2, 12, 14, 15) \quad [3.154]$$

$$\overline{d} = \sum m(1, 4, 7, 10, 15) \quad [3.155]$$

$$\overline{e} = \sum m(1, 3, 4, 5, 7, 9) \quad [3.156]$$

$$\overline{f} = \sum m(1, 2, 3, 7, 13) \quad [3.157]$$

and

$$\overline{g} = \sum m(0, 1, 7, 12) \quad [3.158]$$

Symbole	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
–	1	0	1	0	x	x	x	x	x	x	x
–	1	0	1	1	x	x	x	x	x	x	x
–	1	1	0	0	x	x	x	x	x	x	x
–	1	1	0	1	x	x	x	x	x	x	x
–	1	1	1	0	x	x	x	x	x	x	x
–	1	1	1	1	x	x	x	x	x	x	x

Table 3.46. *Truth table of the BCD-to-7-segment decoder*

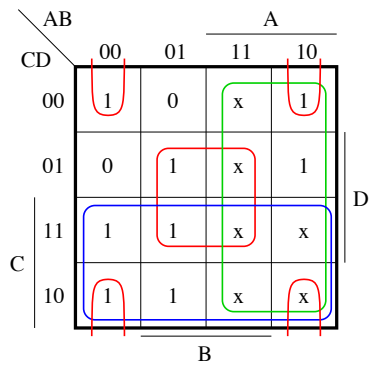


Figure 3.84. *Signal a:*
$a = A + C + B \cdot D + \overline{B} \cdot \overline{D}$
$\overline{a} = B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D$

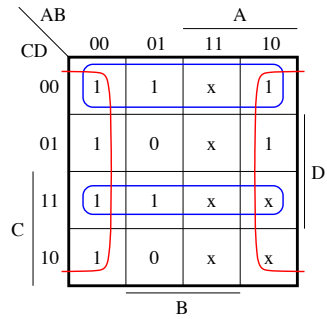


Figure 3.85. *Signal b:*
$b = \overline{B} + C \cdot D + \overline{C} \cdot \overline{D}$
$\overline{b} = B \cdot C \cdot \overline{D} + B \cdot \overline{C} \cdot D$

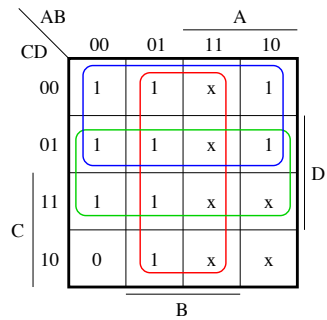


Figure 3.86. *Signal c:*
$c = B + \overline{C} + D$
$\overline{c} = \overline{B} \cdot C \cdot \overline{D}$

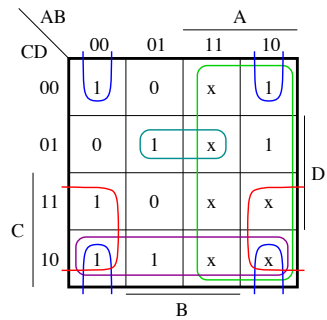


Figure 3.87. *Signal d:*
$d = A + \overline{B} \cdot C + \overline{B} \cdot \overline{D} + C \cdot \overline{D} + B \cdot \overline{C} \cdot D$
$\overline{d} = B \cdot C \cdot D + B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D$

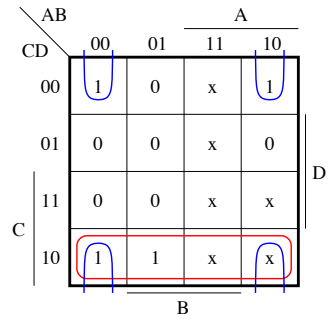


Figure 3.88. *Signal e:*
$e = \overline{B} \cdot \overline{D} + C \cdot \overline{D}$
$\overline{e} = D + B \cdot \overline{C}$

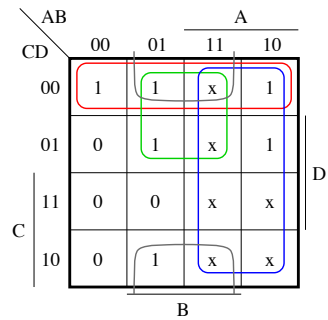


Figure 3.89. *Signal f:*
$f = A + B \cdot \overline{C} + B \cdot \overline{D} + \overline{C} \cdot \overline{D}$
$\overline{f} = \overline{B} \cdot C + C \cdot D + \overline{A} \cdot \overline{B} \cdot D$

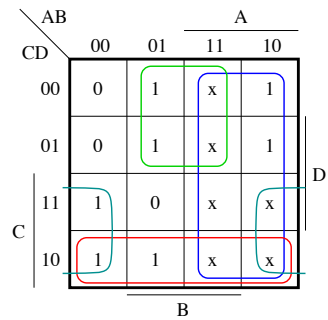


Figure 3.90. *Signal g:*
$g = A + B \cdot \overline{C} + \overline{B} \cdot C + C \cdot \overline{D}$
$\overline{g} = \overline{A} \cdot \overline{B} \cdot \overline{C} + B \cdot C \cdot D$

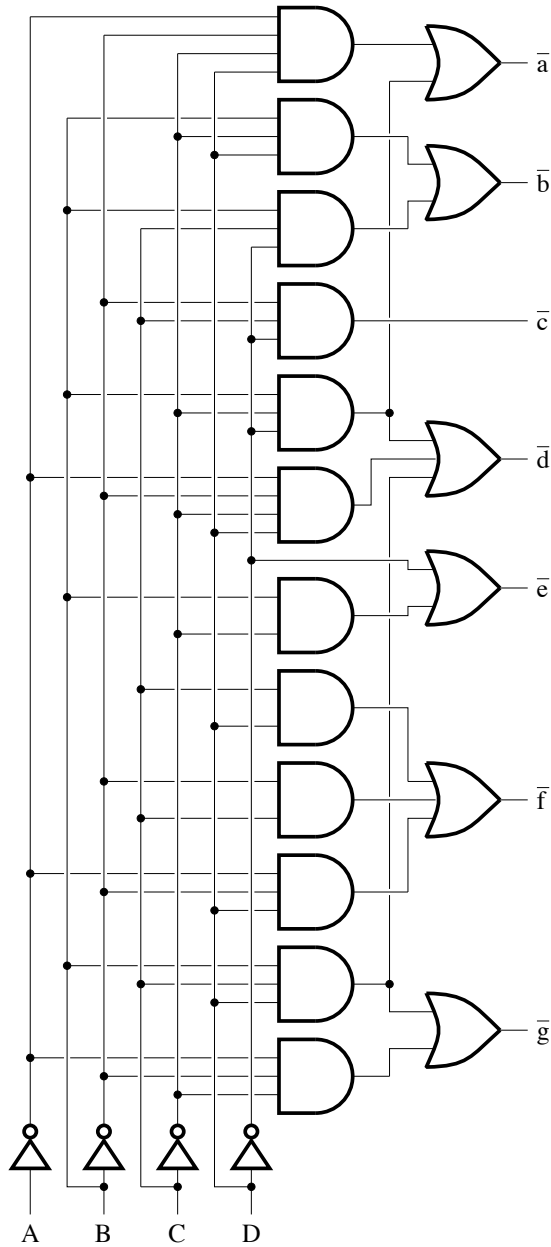


Figure 3.91. *BCD-to-7-segment decoder*

Symbol	A	B	C	D	$\overline{a}$	$\overline{b}$	$\overline{c}$	$\overline{d}$	$\overline{e}$	$\overline{f}$	$\overline{g}$
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0
A	1	0	1	0	0	0	0	1	0	0	0
b	1	0	1	1	1	1	0	0	0	0	0
C	1	1	0	0	0	1	1	0	0	0	1
d	1	1	0	1	1	0	0	0	0	1	0
e	1	1	1	0	0	0	1	0	0	0	0
F	1	1	1	1	0	1	1	1	0	0	0

Table 3.47. *Truth table of the HEX-to-7-segment decoder*

The Karnaugh maps shown in Figures 3.92–3.98 allow for the simplification of the logic equations for the decoder outputs.

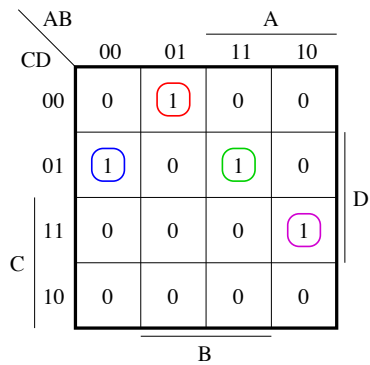


Figure 3.92. *Signal* $\overline{a}$*:*
$\overline{a} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot C \cdot D$

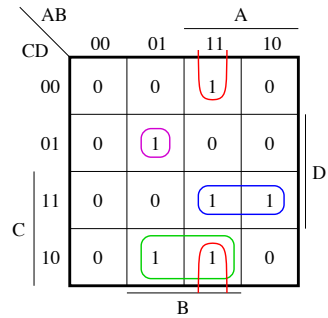


Figure 3.93. *Signal* $\overline{b}$*:* $\overline{b} = A \cdot C \cdot D + A \cdot B \cdot \overline{D} + B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D$

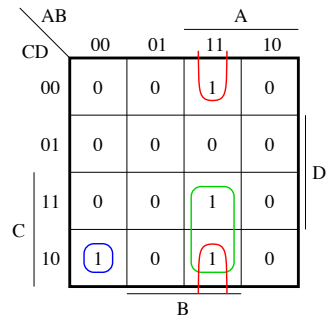


Figure 3.94. *Signal* $\overline{c}$*:* $\overline{c} = A \cdot B \cdot C + A \cdot B \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}$

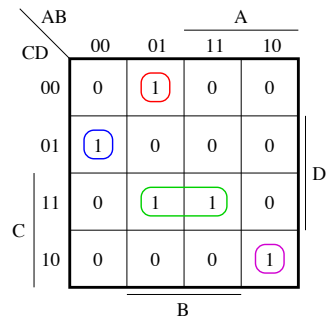


Figure 3.95. *Signal* $\overline{d}$*:*
$\overline{d} = B \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot \overline{D}$

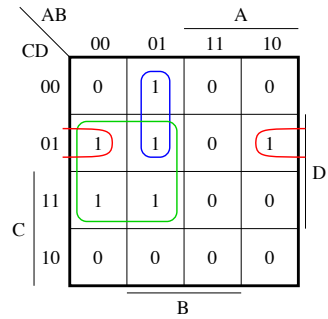


Figure 3.96. *Signal* $\overline{e}$*:*
$\overline{e} = \overline{A} \cdot D + \overline{A} \cdot B \cdot \overline{C} + \overline{B} \cdot \overline{C} \cdot D$

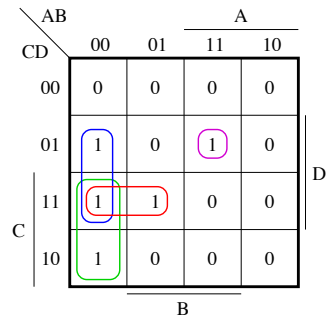


Figure 3.97. *Signal* $\overline{f}$*:* $\overline{f} = \overline{A} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot D + \overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} \cdot D$

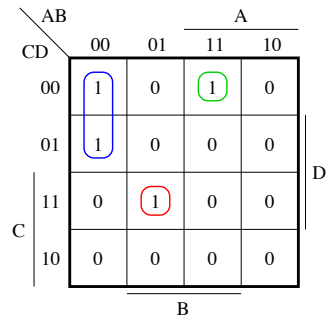


Figure 3.98. *Signal* $\overline{g}$*:*
$\overline{g} = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot C \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D}$

To reduce the number of logic gates, the logic equations of some outputs can be rewritten to bring out the common terms. The logic equation for the output $\overline{b}$ can be put into the form:

$$\overline{b} = A \cdot C \cdot D + A \cdot B \cdot \overline{D} + B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D \quad [3.159]$$

$$= A \cdot C \cdot D + A \cdot B \cdot (C + \overline{C}) \cdot \overline{D} + B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D$$

$$= A \cdot C \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D} + (1 + A) \cdot B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D$$

$$= A \cdot C \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D} + B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot \overline{C} \cdot D \quad [3.160]$$

For the output $\overline{c}$, we have:

$$\overline{c} = A \cdot B \cdot C + A \cdot B \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} \quad [3.161]$$

$$= A \cdot B \cdot C + A \cdot B \cdot (C + \overline{C}) \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}$$

$$= A \cdot B \cdot C \cdot (1 + \overline{D}) + A \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}$$

$$= A \cdot B \cdot C + A \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} \quad [3.162]$$

Another form of the logic equation for the output $\overline{e}$ can be obtained based on the Karnaugh map represented in Figure 3.99. This reveals the common term $\overline{A} \cdot C \cdot D$, instead of the term $\overline{A} \cdot D$, and is written as follows:

$$\overline{e} = \overline{A} \cdot C \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot (D + \overline{D}) + \overline{B} \cdot \overline{C} \cdot D \quad [3.163]$$

$$= \overline{A} \cdot C \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{B} \cdot \overline{C} \cdot D \quad [3.164]$$

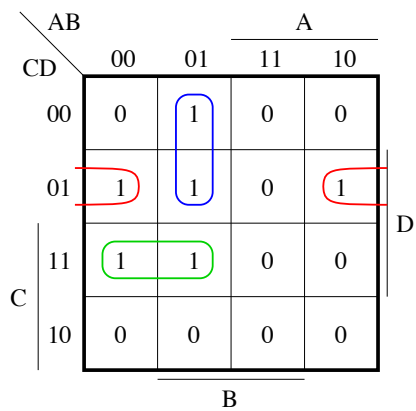


Figure 3.99. *Signal* $\overline{e}$*:*
$\overline{e} = \overline{A} \cdot C \cdot D + \overline{A} \cdot B \cdot \overline{C} + \overline{B} \cdot \overline{C} \cdot D$

The logic equation for the output $\overline{f}$ can be rewritten as:

$$\overline{f} = \overline{A} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot D + \overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} \cdot D \quad [3.165]$$

$$= \overline{A} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot (\overline{C} + C) \cdot D + \overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} \cdot D$$

$$= \overline{A} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot (1 + D) + A \cdot B \cdot \overline{C} \cdot D$$

$$= \overline{A} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} \cdot D \quad [3.166]$$

Figure 3.100 depicts the logic circuit for the HEX-to-7-segment decoder.

SOLUTION 3.17.– Analysis of logic circuits.

1) For the 1-bit barrel shifter, we have:
– inputs: X_4, X_3, X_2, X_1, X_0, X_{-1};
– outputs: Y_3, Y_2, Y_1, Y_0;
– enable signal E;
– signal D:

$$D = \begin{cases} 1 & \text{for a shift to left} \\ 0 & \text{for a shift to right} \end{cases}$$

– signal S to indicate the number of positions to be shifted.

The truth table of the 1-bit barrel shifter is represented in Table 3.48, where:

$$X_{-1} = \begin{cases} 0 & \text{for a shift to the left with insertion of 0} \\ 1 & \text{for a shift to the left with insertion of 1} \\ X_3 & \text{for a rotation to the left} \end{cases}$$

and

$$X_4 = \begin{cases} 0 & \text{for a shift to the right with insertion of 0} \\ 1 & \text{for a shift to the right with insertion of 1} \\ X_0 & \text{for a rotation to the right} \end{cases}$$

E	S	D	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	x	X_3	X_2	X_1	X_0
1	1	0	X_4	X_3	X_2	X_1
1	1	1	X_3	X_2	X_1	X_{-1}

Table 3.48. *Truth table of the barrel shifter*

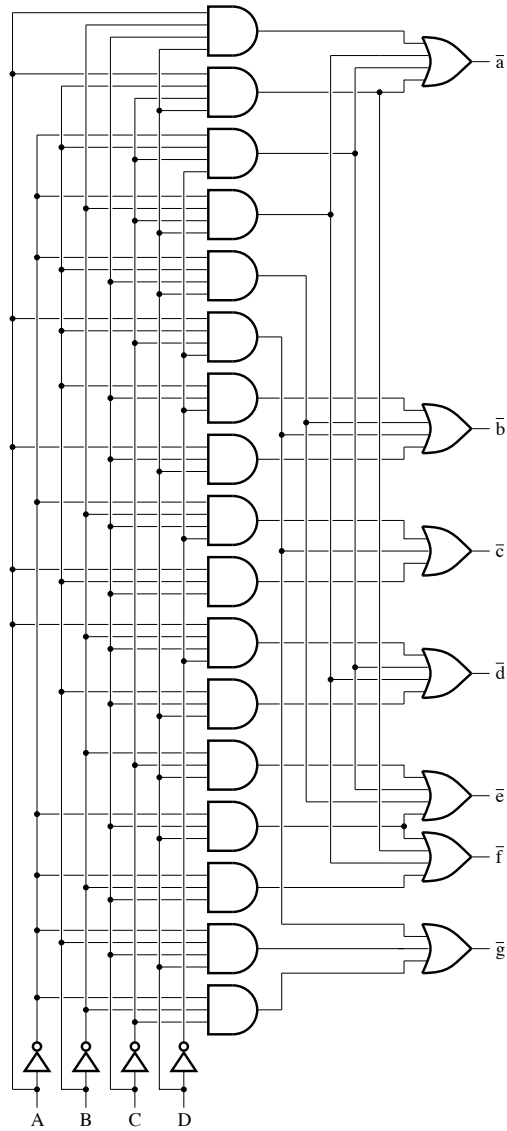


Figure 3.100. *HEX-to-7-segment decoder*

2) The barrel shifter for right-shift operations can be characterized by:

– inputs: D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0;

– outputs: Y_7, Y_6, Y_5, Y_4, Y_3, Y_2, Y_1, Y_0;

– the number of positions to be shifted: S_2, S_1, S_0.

Based on the analysis of the logic circuit, the truth table of the barrel shifter can be constructed as shown in Table 3.49.

S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	0	D_7	D_6	D_5	D_4	D_3	D_2	D_1
0	1	0	0	0	D_7	D_6	D_5	D_4	D_3	D_2
0	1	1	0	0	0	D_7	D_6	D_5	D_4	D_3
1	0	0	0	0	0	0	D_7	D_6	D_5	D_4
1	0	1	0	0	0	0	0	D_7	D_6	D_5
1	1	0	0	0	0	0	0	0	D_7	D_6
1	1	1	0	0	0	0	0	0	0	D_7

Table 3.49. *Truth table of the barrel shifter for right-shift operations*

4

Systematic Methods for the Simplification of Logic Functions

4.1. Introduction

Manipulating Karnaugh maps can prove a difficult task for logic functions of more than six variables. Most often, systematic methods of simplification such as the Quine–McCluskey method or the iterated consensus method are used to find minimized forms for functions with a large number of variables. These methods are especially useful because they can be converted into algorithms or computer-aided design software. A logic function of n variables can have up to 2^n minterms and $3^n/n$ prime implicants. The implementation of methods that require the enumeration of all the minterms and the determination of the prime implicants seems to be limited by the calculation and storage capacity that is excessively high when the number of variables increases. For this reason, functions with a large number of variables are minimized using iterative heuristic methods such as the Espresso algorithm.

4.2. Definitions and reminders

Consider the following logic function:

$$F(A,B,C,D) = \sum m(2,3,4,5,6,7,9,11,12,13) \quad [4.1]$$

The minimized form of the function F can be obtained using the Karnaugh map shown in Figure 4.1(a). That is:

$$F(A,B,C,D) = \overline{A} \cdot C + B \cdot \overline{C} + A \cdot \overline{B} \cdot D \quad [4.2]$$

Digital Electronics 1: *Combinational Logic Circuits*, First Edition. Tertulien Ndjountche.
© ISTE Ltd 2016. Published by ISTE Ltd and John Wiley & Sons, Inc.

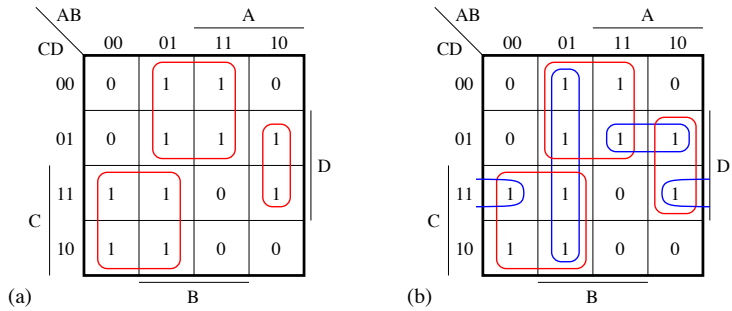


Figure 4.1. *Karnaugh map for* $F(A, B, C, D)$

4.2.1. *Definitions*

Any single logic state 1 or group of 1s that can be combined on a Karnaugh map for a logic function represents a term (or product of variables) that is called an *implicant*.

In general, for a logic function F of n variables, a minterm m_i is an implicant if, for all combination of logic levels of the n variables for which m_i takes the logic state 1, F is also at logic state 1.

A *prime implicant* for a logic function is a term that cannot be combined with another term to eliminate a variable.

A prime implicant is said to be essential if it is the only one to cover (or include) one or more minterms.

4.2.2. *Minimization principle of a logic function*

The objective of minimizing a logic function is to determine the smallest number of prime implicants that, together, cover all the minterms for this function. The minimized form of a logic function contains all the essential prime implicants.

The prime implicants and the essential prime implicants for a logic function can be determined from a Karnaugh map. A single 1 represents a prime implicant if there is no other 1 neighboring it. Two neighboring 1s represent a prime implicant if they cannot be contained in a group of four 1s, four neighboring 1s form a prime implicant if they cannot be included in a group of eight 1s and so on.

In the specific case of the function that is defined by equation [4.1], the Karnaugh map shown in Figure 4.1(b) can be used to obtain the following prime implicants:

$$\overline{A} \cdot C,\ B \cdot \overline{C},\ A \cdot \overline{B} \cdot D,\ \overline{A} \cdot B,\ A \cdot \overline{C} \cdot D \text{ and } \quad \overline{B} \cdot C \cdot D.$$

The essential prime implicants are $\overline{A} \cdot C$ and $B \cdot \overline{C}$.

In general, as each minterm is covered by at least one of the prime implicants, a logic function is equal to the sum of its prime implicants. Thus, the function F can be expressed in the form:

$$F(A, B, C, D) = \overline{A} \cdot B + \overline{A} \cdot C + B \cdot \overline{C} + A \cdot \overline{B} \cdot D + A \cdot \overline{C} \cdot D + \overline{B} \cdot C \cdot D \quad [4.3]$$

Equation [4.3] does not have a minimal number of terms and it is, consequently, not the minimized sum-of-products form. However, each of its terms has a minimum number of variables.

4.3. Karnaugh maps

The use of a Karnaugh map can be extended to logic functions of more than four variables. However, in practice, the manipulation of a Karnaugh map only proves to be easy for up to six variables.

To determine the minimized sum-of-products or product-of-sums form for a given logic function using a Karnaugh map, it is necessary to:

1) express the function in the form of the sum of its minterms or maxterms;

2) place a 1 (or 0) in the appropriate cell for each minterm (or maxterm);

3) cover all the 1s (or 0s) using a minimum number of the largest possible loops that encompass 2^p cells, p being an integer, while also ensuring that each 1 (or 0) is part of at least one loop. It is preferable to identify the possible loops by beginning with the cells that can be grouped in only one way;

4) form the simplified expression for the logic function by summing (or multiplying) the terms obtained for the different loops. It must be noted that the term associated with a loop enclosing 2^p cells is obtained by eliminating the p variables that change logic state.

4.3.1. *Function of five variables*

In the case of a logic function of five variables, the Karnaugh map consists of 32 (or 2^5) cells bearing the numbers from 0 to 31.

The direct approach to constructing a Karnaugh map consists of dividing the variables into two groups of terms that are ordered horizontally and vertically according to reflected binary code (or Gray code) to mark each cell. The five-variable Karnaugh map obtained in this manner has two symmetrical axes, as shown in

Figure 4.2. The first column and the eighth column are adjacent as are the second and seventh columns, the third and sixth columns, and the fourth and fifth columns.

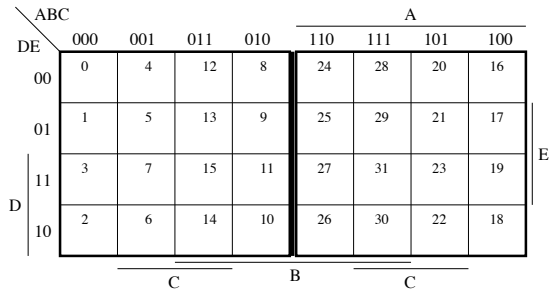


Figure 4.2. *Five-variable Karnaugh map: symmetrical presentation*

A five-variable Karnaugh map can also be constructed based on a layered three-dimensional representation of two diagrams of four variables as given in Figure 4.3(a), or based on a bidimensional and asymmetrical representation as illustrated in Figure 4.3(b). In each case, we have one map for $\overline{A}$ $(A = 0)$ and another for A $(A = 1)$. In general, two cells are adjacent when they correspond to minterms that only differ by one variable. Thus, each cell in one of the maps is adjacent to a corresponding cell on the other map.

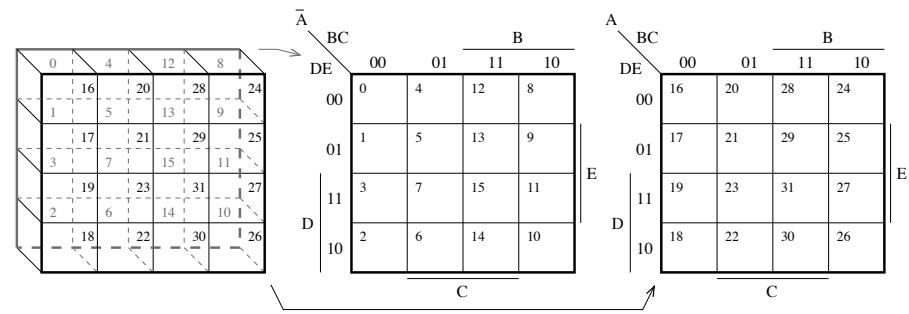


Figure 4.3. *Five-variable Karnaugh map:*
a) three-dimensional representation and b) bidimensional and non-symmetrical representation

Figures 4.4 and 4.5 give some examples of loops, representing the following terms:

– loop 1: $\overline{B} \cdot C \cdot \overline{D} \cdot E$;

– loop 2: $B \cdot C \cdot E$;

– loop 3: $\overline{C} \cdot \overline{E}$.

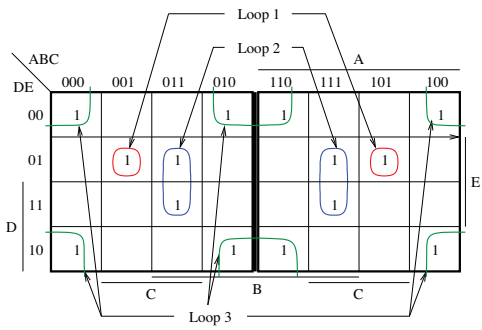


Figure 4.4. *Example of loops in the case of a symmetrical map*

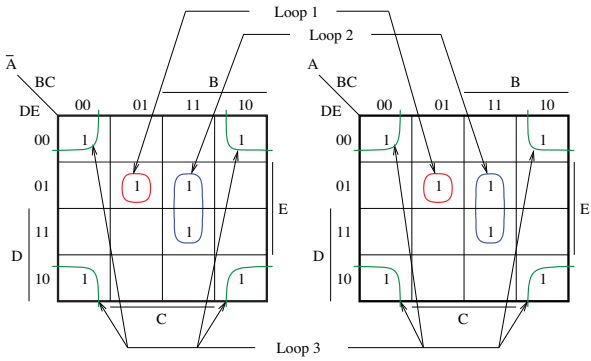


Figure 4.5. *Example of loops in the case of an asymmetrical map*

It should be noted that only the loops that encircle 2^p cells, p being an integer, are valid and each loop must be symmetrical with respect to any axis that divides it.

4.3.2. *Function of six variables*

A six-variable Karnaugh map has 64 (or 2^6) cells, numbered from 0 to 63. It can be constructed either using Gray code to identify the cells, as shown in Figure 4.6, or based on a stacked three-dimensional structure of four maps of four variables (see Figure 4.7). In the latter case, in addition to the possibility of locating adjacent cells

on a horizontal or vertical plane, or at the ends of the same plane, cells on upper and lower planes can be considered adjacent.

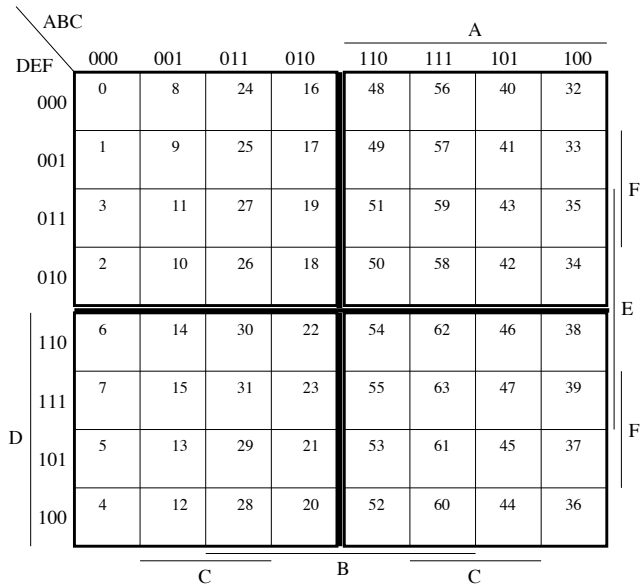


Figure 4.6. *Six-variable Karnaugh map: symmetrical structure*

4.3.3. *Karnaugh map with entered variable*

A Karnaugh map with an entered variable makes it possible to manipulate a logic function that has more variables than in the map. It is constructed by entering, in addition to 1 and 0, the variables in cells of a Karnaugh map. It is most useful in cases where some variables appear less frequently in a given logic function.

By also entering variables in an n-variable Karnaugh map to represent a function of N variables, each cell becomes the equivalent of a submap that can cover, for $N > n$, 2^{N-n} possible minterms or maxterms.

The minimized sum-of-products (or product-of-sum forms) form of a logic function can, thus, be determined as follows:

1) form the loops by grouping the entered variables of the same type or those whose logic adjacencies can permit minimum cover and adjacent cells containing the logic state 1 (or 0), or representations of don't care states. Obtain the simplified term for each loop, considering the 1s (or 0s) as indifferent states;

2) form loops that encircle only adjacent cells containing the 1s (or 0s) that are not covered or not completely covered and those representing don't care states, if any. Obtain the simplified term for each loop, by eliminating each variable that is simultaneously complemented and non-complemented;

3) write the simplified equation for the logic function by summing (or by multiplying) the terms obtained for all the loops.

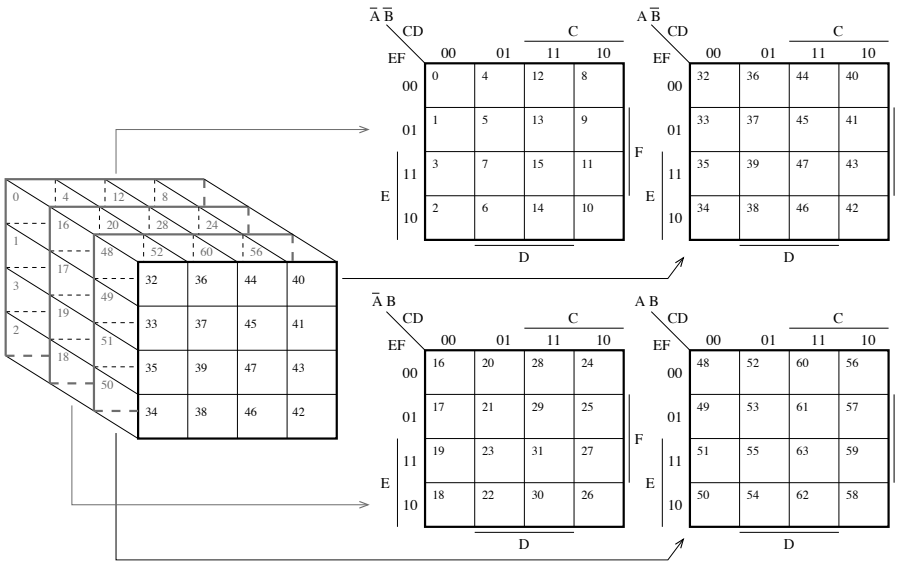


Figure 4.7. *Six-variable Karnaugh map: stacked structure*

EXAMPLE 4.1.– Let Z be a logic function of five variables characterized by the truth table shown in Table 4.1, where x represents a don't care state:

– express the function Z in the canonical sum-of-products form;

– determine the minimized form of the function Z.

According to the truth table, the logic function Z can be written as:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}(x+E) \\ & + \overline{A}\cdot B\cdot\overline{C}\cdot D\cdot E + A\cdot\overline{B}\cdot\overline{C}(x+D+E) + A\cdot\overline{B}\cdot C \\ & + A\cdot B\cdot\overline{C}\cdot E \end{aligned} \qquad [4.4]$$

A	B	C	Z
0	0	0	$D \cdot E$
0	0	1	$\overline{D}(x + E)$
0	1	0	$D \cdot E$
0	1	1	0
1	0	0	$x + D + E$
1	0	1	1
1	1	0	E
1	1	1	0

Table 4.1. *Truth table*

Transforming this last expression so that only minterms can appear, we get:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D \cdot E + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} \cdot \overline{E} \cdot x + \\ & \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} \cdot E(1 + x) + \overline{A} \cdot B \cdot \overline{C} \cdot D \cdot E + \\ & A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot \overline{E} \cdot x + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot E(1 + x) + \\ & A \cdot \overline{B} \cdot \overline{C} \cdot D \cdot \overline{E}(1 + x) + A \cdot \overline{B} \cdot \overline{C} \cdot D \cdot E(1 + x) + \\ & A \cdot \overline{B} \cdot C \cdot \overline{D} \cdot \overline{E} + A \cdot \overline{B} \cdot C \cdot \overline{D} \cdot E + A \cdot \overline{B} \cdot C \cdot D \cdot \overline{E} + \\ & A \cdot \overline{B} \cdot C \cdot D \cdot E + A \cdot B \cdot \overline{C} \cdot D \cdot E + A \cdot B \cdot \overline{C} \cdot \overline{D} \cdot E \end{aligned} \quad [4.5]$$

Finally, the function Z can be expressed in the following canonical form:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \sum m(3,5,11,17,18,19,20,21,22,23,25,27) \\ & + \sum x(4,16) \end{aligned} \quad [4.6]$$

Figure 4.8(a) depicts the Karnaugh map constructed from the truth table of function Z. Choosing x = 1 makes it possible to reduce the complexity of the terms entered in the Karnaugh map as shown in Figure 4.8(b). Because

$$1 = D + \overline{D} \quad [4.7]$$

$$= (D + \overline{D})(E + \overline{E}) = D \cdot E + \overline{D} \cdot E + D \cdot \overline{E} + \overline{D} \cdot \overline{E} \quad [4.8]$$

and

$$E = (D + \overline{D})E = D \cdot E + \overline{D} \cdot E \quad [4.9]$$

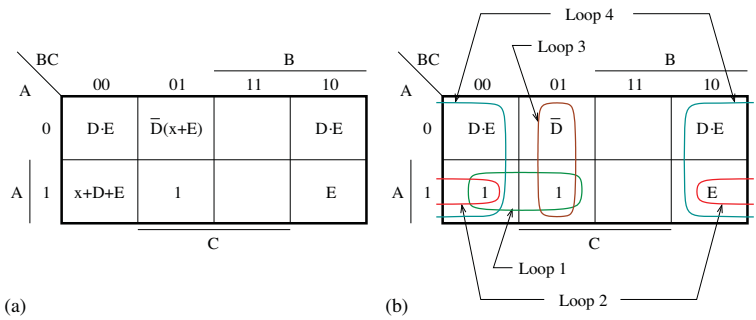


Figure 4.8. *a) Karnaugh map with two entered variables (x being a don't care state); b) Karnaugh map when x = 1*

Loop 2 is assumed to encircle the term E, loop 3 encircles the term $\overline{D}$ and loop 4 the term $D \cdot E$. It is, thus, necessary to cover the remaining terms by forming loop 1 that encircles the two adjacent 1s. The reduced term corresponding to each of the loops is obtained as follows:

$$\text{Loop 1} \rightarrow A \cdot \overline{B}$$
$$\text{Loop 2} \rightarrow A \cdot \overline{C} \cdot E$$
$$\text{Loop 3} \rightarrow \overline{B} \cdot C \cdot \overline{D}$$
$$\text{Loop 4} \rightarrow \overline{C} \cdot D \cdot E$$

In the minimized sum-of-products form, the function Z can then be written as:

$$Z(A, B, C, D, E) = A \cdot \overline{B} + A \cdot \overline{C} \cdot E + \overline{B} \cdot C \cdot \overline{D} + \overline{C} \cdot D \cdot E \qquad [4.10]$$

Figure 4.9 depicts the Karnaugh map (x = 1) to determine the minimized product-of-sums form. The terms obtained for the different loops are as follows:

$$\text{Loop 1} \rightarrow \overline{B} + \overline{C}$$
$$\text{Loop 2} \rightarrow \overline{B} + E$$
$$\text{Loop 3} \rightarrow A + C + D$$
$$\text{Loop 4} \rightarrow A + \overline{C} + \overline{D}$$
$$\text{Loop 5} \rightarrow A + C + E$$

Thus, the minimized product-of-sums form is given by:

$$Z(A, B, C, D, E) = (\overline{B} + \overline{C})(\overline{B} + E)(A + C + D)(A + \overline{C} + \overline{D})(A + C + E) \qquad [4.11]$$

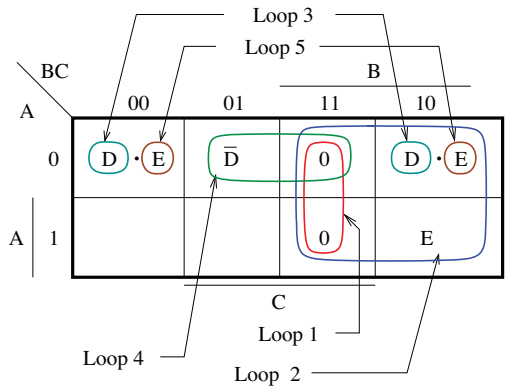


Figure 4.9. *Karnaugh map (x= 1) to determine the minimized product-of-sums form*

NOTE 4.1.– To simplify an incompletely defined function using a Karnaugh map with entered variables, the don't care state must be considered as an entered variable.

EXAMPLE 4.2.– Let us consider the following logic function Z of six variables:

$$Z(A,B,C,D,E,F) = \sum m(4,6,8,9,10,11,12,13,14,15,20,22,26,27,30,31,32,33,34,35,36,38,39,52,54,56,57,60,61) \quad [4.12]$$

Determine the minimized form for the function Z.

The first step consists of entering, in addition to the 1s, two variables (E and F) in a Kanraugh map for four variables. We proceed by associating the combination of minterms with the cells of the Karnaugh map as follows:

cell 1: $m_4 + m_6 = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D \cdot \overline{F}$
cell 2: $m_8 + m_9 + m_{10} + m_{11} = \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}$
cell 3: $m_{12} + m_{13} + m_{14} + m_{15} = \overline{A} \cdot \overline{B} \cdot C \cdot D$
cell 5: $m_{20} + m_{22} = \overline{A} \cdot B \cdot \overline{C} \cdot D \cdot \overline{F}$
cell 6: $m_{26} + m_{27} = \overline{A} \cdot B \cdot C \cdot \overline{D} \cdot E$
cell 7: $m_{30} + m_{31} = \overline{A} \cdot B \cdot C \cdot D \cdot E$
cell 8: $m_{32} + m_{33} + m_{34} + m_{35} = A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$
cell 9: $m_{36} + m_{38} + m_{39} = A \cdot \overline{B} \cdot \overline{C} \cdot D(E + \overline{F})$
cell 13: $m_{52} + m_{54} = A \cdot B \cdot \overline{C} \cdot D \cdot \overline{F}$
cell 14: $m_{56} + m_{57} = A \cdot B \cdot C \cdot \overline{D} \cdot \overline{E}$
cell 15: $m_{60} + m_{61} = A \cdot B \cdot C \cdot D \cdot \overline{E}$

We next encircle the adjacent cells containing 1s or the same variable to form loops as illustrated in Figure 4.10. In this way, we can obtain a term of the following form for each loop:

Loop 1 $\rightarrow \overline{A} \cdot \overline{B} \cdot C$
Loop 2 $\rightarrow \overline{A} \cdot C \cdot E$
Loop 3 $\rightarrow \overline{C} \cdot D \cdot \overline{F}$
Loop 4 $\rightarrow A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$
Loop 5 $\rightarrow A \cdot B \cdot C \cdot \overline{E}$
Loop 6 $\rightarrow A \cdot \overline{B} \cdot \overline{C} \cdot E$

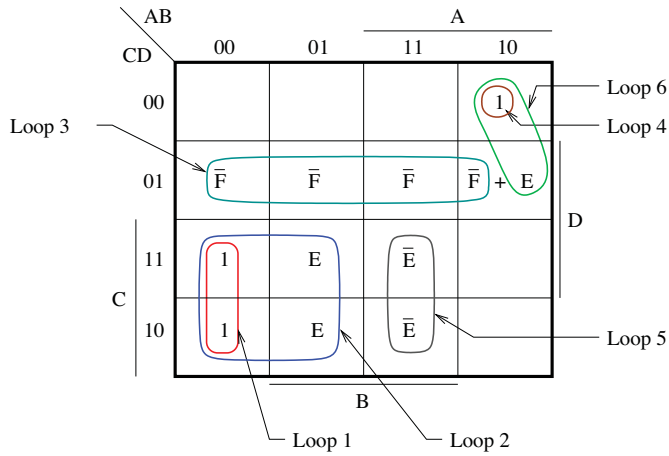


Figure 4.10. *Karnaugh map with two entered variables to determine the minimized sum-of-products form*

The function Z is thus expressed in the following minimized sum-of-products form:

$$Z(A,B,C,D,E,F) = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot C \cdot E + \overline{C} \cdot D \cdot \overline{F} + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot C \cdot \overline{E} + A \cdot \overline{B} \cdot \overline{C} \cdot E \quad [4.13]$$

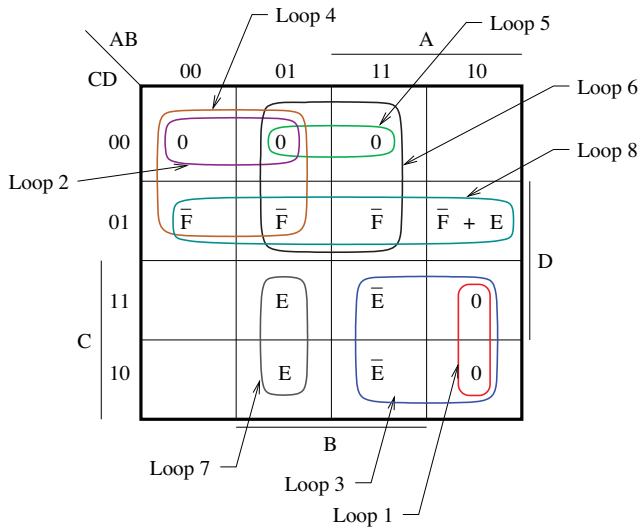


Figure 4.11. *Karnaugh map with two entered variables to determine the minimized product-of-sums forms*

The minimized product-of-sums form can be determined based on the Karnaugh map shown in Figure 4.11. Forming loops that can encircle the entered variables or the 0s, we can obtain:

$$\begin{aligned}
&\text{Loop 1} \rightarrow \overline{A} + B + \overline{C}\\
&\text{Loop 2} \rightarrow A + C + D\\
&\text{Loop 3} \rightarrow \overline{A} + \overline{C} + \overline{E}\\
&\text{Loop 4} \rightarrow A + C + \overline{F}\\
&\text{Loop 5} \rightarrow \overline{B} + C + D\\
&\text{Loop 6} \rightarrow \overline{B} + C + \overline{F}\\
&\text{Loop 7} \rightarrow A + \overline{B} + \overline{C} + E\\
&\text{Loop 8} \rightarrow C + \overline{D} + E + \overline{F}
\end{aligned}$$

Finally, the minimized product-of-sums form can be written as follows:

$$\begin{aligned}
Z(A,B,C,D,E,F) = &(\overline{A} + B + \overline{C})(A + C + D)(\overline{A} + \overline{C} + \overline{E})(A + C + \overline{F})\\
&(\overline{B} + C + D)(\overline{B} + C + \overline{F})(A + \overline{B} + \overline{C} + E)\\
&(C + \overline{D} + E + \overline{F})
\end{aligned} \qquad [4.14]$$

4.3.4. *Applications*

EXAMPLE 4.3.– Let us consider the following logic function of five variables:

$$Z(A,B,C,D,E) = \sum m(0,1,4,5,6,12,14,16,20,22,25,28,30,31) \quad [4.15]$$

Determine the minimized sum-of-products form for the function Z.

Figure 4.12 depicts the Karnaugh map for the function Z. The loop encircling the minterms 4, 6, 12, 14, 20, 22, 28 and 30 yields the term $C \cdot \overline{E}$; the loop encircling the minterms 0, 1, 4, and 5 yields $\overline{A} \cdot \overline{B} \cdot \overline{D}$; the loop encircling the minterms 0, 4, 16 and 20 yields $\overline{B} \cdot \overline{D} \cdot \overline{E}$; the loop encircling the minterms 30 and 31 yields $A \cdot B \cdot C \cdot D$; and the loop encircling 25 yields $A \cdot B \cdot \overline{C} \cdot \overline{D} \cdot E$. Thus:

$$Z(A,B,C,D,E) = C \cdot \overline{E} + \overline{A} \cdot \overline{B} \cdot \overline{D} + \overline{B} \cdot \overline{D} \cdot \overline{E} + A \cdot B \cdot C \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D} \cdot E \quad [4.16]$$

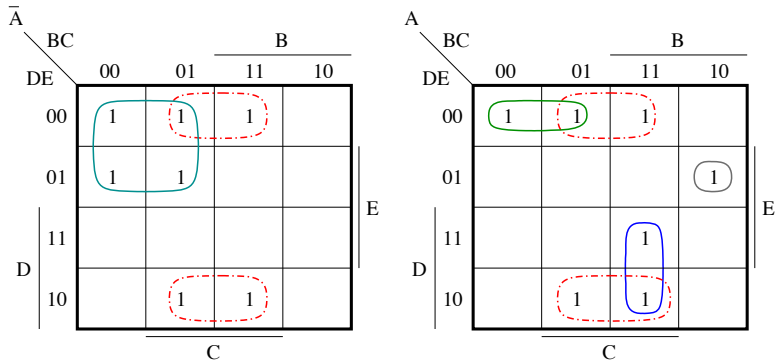


Figure 4.12. *Karnaugh map*

EXAMPLE 4.4.– A function of five variables is defined by:

$$Z(A,B,C,D,E) = \sum m(3,7,11,12,13,14,15,16,18) + \sum x(24,25,26,27,28,29,30,31) \quad [4.17]$$

Determine the minimized sum-of-products for Z using a five-variable Karnaugh map and choosing to enter a variable in a four-variable Karnaugh map.

– grouping the adjacent cells of the five-variable Karnaugh map shown in Figure 4.13(a), we can obtain the following different terms for the simplified expression of Z:

Loop encircling cells 12, 13, 14, 15, 28, 29, 30, and 31 $\rightarrow B \cdot C$
Loop encircling cells 16, 18, 24 and 26 $\rightarrow A \cdot \overline{C} \cdot \overline{E}$
Loop encircling cells 3, 7, 11 and 15 $\rightarrow \overline{A} \cdot D \cdot E$

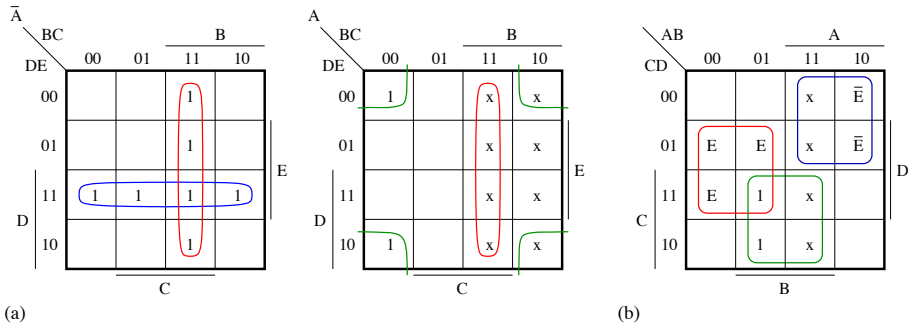


Figure 4.13. *a) Five-variable Karnaugh map and b) Karnaugh map with an entered variable*

The minimal sum-of-products can then be written as:

$$Z(A,B,C,D,E) = B \cdot C + A \cdot \overline{C} \cdot \overline{E} + \overline{A} \cdot D \cdot E \qquad [4.18]$$

– the function Z can be simplified by entering one of the variables in the four-variable Karnaugh map. To fill up the Karnaugh map, the terms of the logic function are associated with the cells in the following manner:

cell 1: $m_3 = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D \cdot E$ cell 3: $m_7 = \overline{A} \cdot \overline{B} \cdot C \cdot D \cdot E$
cell 5: $m_3 = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D \cdot E$ cell 6: $m_{12} + m_{13} = \overline{A} \cdot B \cdot C \cdot \overline{D}$
cell 7: $m_{14} + m_{15} = \overline{A} \cdot B \cdot C \cdot D$ cell 8: $m_{16} = A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot \overline{E}$
cell 9: $m_{18} = A \cdot \overline{B} \cdot \overline{C} \cdot D \cdot \overline{E}$
cell 12: $d_{24} + d_{25} = A \cdot B \cdot \overline{C} \cdot \overline{D}$ cell 13: $d_{26} + d_{27} = A \cdot B \cdot \overline{C} \cdot D$
cell 14: $d_{28} + d_{29} = A \cdot B \cdot C \cdot \overline{D}$ cell 15: $d_{30} + d_{31} = A \cdot B \cdot C \cdot D$

The different loops obtained by grouping the cells in the map shown in Figure 4.13(b) yield the following products:

Loop encircling cells 7, 6, 14 and 15 $\rightarrow B \cdot C$
Loop encircling cells 8, 9, 12 and 13 $\rightarrow A \cdot \overline{C} \cdot \overline{E}$
Loop encircling cells 1, 3, 5 and 7 $\rightarrow \overline{A} \cdot D \cdot E$

We can obtain the same expression as above for the function Z, that is:

$$Z(A, B, C, D, E) = B \cdot C + A \cdot \overline{C} \cdot \overline{E} + \overline{A} \cdot D \cdot E \qquad [4.19]$$

EXAMPLE 4.5.– Using a six-variable Karnaugh map first and then entering two variables in a four-variable Karnaugh map, simplify the following logic function:

$$Z(A, B, C, D, E, F) = \sum m(0, 2, 4, 6, 8, 10, 12, 14, 16, 20, 23, 32, 34, 36, 38, 40, 42, 44, 45, 46, 49, 51, 53, 54, 55, 57, 59, 60, 61, 62, 63) \qquad [4.20]$$

– The Karnaugh map is filled in by inserting a 1 in each cell that corresponds to a minterm of the logic function. The six-variable Karnaugh map for the logic function Z is given in Figure 4.14. The cells in the map can be grouped in the following manner:

Loop encircling cells 0, 2, 4, 6, 8, 10, 12, 14, 32, 34, 36, 38, 40, 42, 44 and 46 $\rightarrow \overline{B} \cdot \overline{F}$
Loop encircling cells 49, 51, 53, 55, 57, 59, 61 and 63 $\rightarrow A \cdot B \cdot F$
Loop encircling cells 0, 4, 16 and 20 $\rightarrow \overline{A} \cdot \overline{C} \cdot \overline{E} \cdot \overline{F}$
Loop encircling cells 23 and 55 $\rightarrow B \cdot \overline{C} \cdot D \cdot E \cdot F$
Loop encircling cells 38, 46, 54 and 62 $\rightarrow A \cdot D \cdot E \cdot \overline{F}$
Loop encircling cells 44, 45, 60 and 61 $\rightarrow A \cdot C \cdot D \cdot \overline{E}$

The simplified expression of Z is then given by:

$$Z(A, B, C, D, E, F) = \overline{B} \cdot \overline{F} + A \cdot B \cdot F + A \cdot C \cdot D \cdot \overline{E} + A \cdot D \cdot E \cdot \overline{F} + \overline{A} \cdot \overline{C} \cdot \overline{E} \cdot \overline{F} + B \cdot \overline{C} \cdot D \cdot E \cdot F \qquad [4.21]$$

– Minterms of the function Z are assigned to cells in the four-variable Karnaugh map as follows:

cell 0: $m_0 + m_2 = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot \overline{F}$
cell 1: $m_4 + m_6 = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D \cdot \overline{F}$
cell 2: $m_8 + m_{10} = \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} \cdot \overline{F}$
cell 3: $m_{12} + m_{14} = \overline{A} \cdot \overline{B} \cdot C \cdot D \cdot \overline{F}$
cell 4: $m_{16} = \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} \cdot \overline{E} \cdot \overline{F}$
cell 5: $m_{20} + m_{23} = \overline{A} \cdot B \cdot \overline{C} \cdot D(\overline{E} \cdot \overline{F} + E \cdot F)$
cell 8: $m_{32} + m_{34} = A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot \overline{F}$
cell 9: $m_{36} + m_{38} = A \cdot \overline{B} \cdot \overline{C} \cdot D \cdot \overline{F}$
cell 10: $m_{40} + m_{42} = A \cdot \overline{B} \cdot C \cdot \overline{D} \cdot \overline{F}$
cell 11: $m_{44} + m_{45} + m_{46} = A \cdot \overline{B} \cdot C \cdot D(\overline{E} + \overline{F})$
cell 12: $m_{49} + m_{51} = A \cdot B \cdot \overline{C} \cdot \overline{D} \cdot F$
cell 13: $m_{53} + m_{54} + m_{55} = A \cdot B \cdot \overline{C} \cdot D(E + F)$
cell 14: $m_{57} + m_{59} = A \cdot B \cdot C \cdot \overline{D} \cdot F$
cell 15: $m_{60} + m_{61} + m_{62} + m_{63} = A \cdot B \cdot C \cdot D$

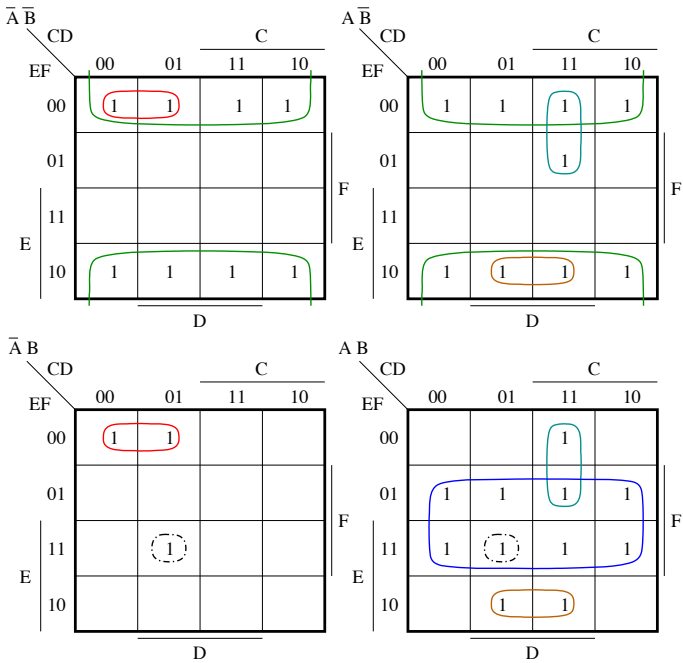


Figure 4.14. *Six-variable Karnaugh map*

The products obtained from the grouping carried out in the Karnaugh map shown in Figure 4.15 are written as follows:

Loop 1 $\rightarrow \overline{B} \cdot \overline{F}$
Loop 2 $\rightarrow A \cdot B \cdot F$
Loop 3 $\rightarrow \overline{A} \cdot \overline{C} \cdot \overline{E} \cdot \overline{F}$
Loop 4 $\rightarrow B \cdot \overline{C} \cdot D \cdot E \cdot F$
Loop 5 $\rightarrow A \cdot D \cdot E \cdot \overline{F}$
Loop 6 $\rightarrow A \cdot C \cdot D \cdot \overline{E}$

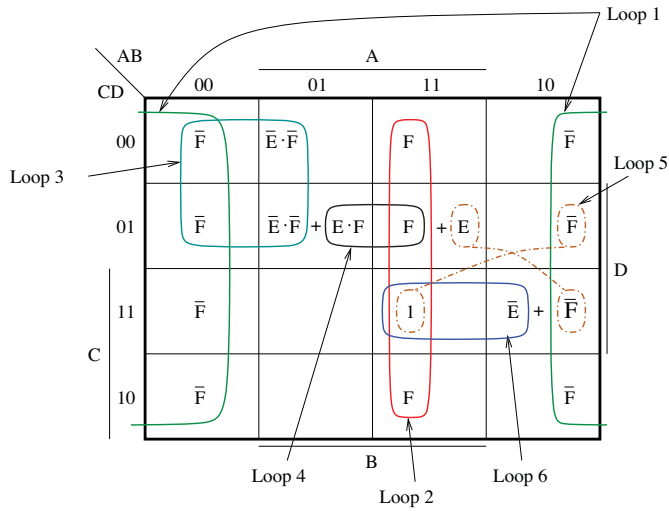


Figure 4.15. *Karnaugh map with entered variables*

The elimination of the variables permitted by loops 3, 4 and 5 are explained by the fact that:

$$\overline{F} = (E + \overline{E})\overline{F} = E \cdot \overline{F} + \overline{E} \cdot \overline{F} \qquad [4.22]$$

and

$$E + F = (E + F)(F + \overline{F}) = F + E \cdot F + E \cdot \overline{F} \qquad [4.23]$$

As before, we obtain the simplified equation for the function Z that is equal to:

$$Z(A, B, C, D, E, F) = \overline{B} \cdot \overline{F} + A \cdot B \cdot F + A \cdot C \cdot D \cdot \overline{E} + A \cdot D \cdot E \cdot \overline{F} + \overline{A} \cdot \overline{C} \cdot \overline{E} \cdot \overline{F} + B \cdot \overline{C} \cdot D \cdot E \cdot F \qquad [4.24]$$

4.3.5. *Representation based on the XOR and AND operators*

For some logic functions, it may be necessary to use representation based on the XOR and AND gates instead of using sum-of-products representations. This is especially the case if the objective is to minimize the total number of logic gates and the interconnect complexity.

The Karnaugh map depicted in Figure 4.16 gives a representation for each of the three logic functions used as examples. Three types of loops can be identified on the map, including minterms that are diagonal, one position apart from each other, or adjacent. The logic expressions that can be associated with the different loops can be written as follows:

– loop 1: $F_1 = A \cdot B \cdot \overline{C} \cdot D \cdot X + A \cdot \overline{B} \cdot C \cdot D \cdot X = A \cdot D \cdot X(B \oplus C)$;

– loop 2: $F_2 = \overline{A} \cdot \overline{B} \cdot C \cdot D \cdot Y + A \cdot B \cdot C \cdot D \cdot Y = (\overline{A \oplus B})C \cdot D \cdot Y$;

– loop 3: $F_3 = \overline{A} \cdot B \cdot \overline{C} \cdot D \cdot \overline{Z} + \overline{A} \cdot B \cdot C \cdot D \cdot Z = \overline{A} \cdot B \cdot D(\overline{C \oplus Z})$.

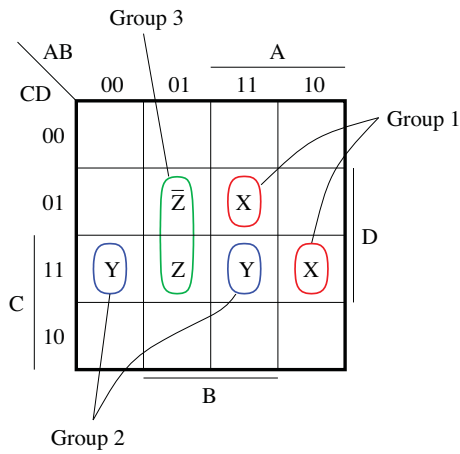


Figure 4.16. *Karnaugh map with entered variables*

We can see that each expression is made more compact using the XOR logic function.

4.4. Systematic methods for simplification

The Karnaugh method is appropriate for the simplification of logic functions with a small number of variables. As the number of variables increases, systematic

procedures or algorithms are used for simplification. The implementation of these algorithms takes place in two steps. The first step is the determination of the prime implicants. The second step consists of constituting the set of terms that make up each minimized logic expression.

4.4.1. *Determination of prime implicants*

Among the methods that are commonly employed to determine the prime implicants of a logic function are the Quine–McCluskey method and the iterated consensus method.

4.4.1.1. *Quine–McCluskey method*

In order to determine all the prime implicants of a logic function, all the minterms must be compared and combined two by two so as to eliminate those that are not necessary to cover the function.

To reduce the number of comparisons, the minterms of the canonical decomposition are classified based on the number of 1s that appear in the representation, before they are inscribed in the first column of a table.

Each minterm of the group k is logically combined with each minterm of group $k+1$ using the logic identity $X \cdot Y + \overline{X} \cdot Y = Y$. When a single variable can be eliminated, the result is consigned to the second-column group whose number is equal to the number of 1s it possesses.

With the help of the ✓ symbol the two terms in question can be identified to show that they can no longer be included in the sum-of-products expression.

If a term has already been obtained, it is no longer reinscribed in the table. On the contrary, using the ✓ symbol identifies the two terms that have been combined.

In the representation of a term, the - symbol replaces the eliminated variable.

As before, we then proceed with the combination of the terms in groups k and $k+1$ in the second column. However, only those terms that have the - symbol in the same position can be combined.

If necessary, the operation of combining the terms of the groups k and $k+1$ can be repeated for the next column until it is no longer possible to combine terms.

The terms that can no longer be combined and that are represented using the $\star$ symbols are the prime implicants.

NOTE 4.2.– Only those terms that belong to two contiguous groups or that are differentiated by a single and identical variable can be combined.

It is not necessary to combine terms that belong to two non-contiguous groups as these are differentiated by at least two variables. For the same reason, we do not try to combine terms in the same group.

In the example of the logic function:

$$F(A,B,C,D) = \sum m(2,3,4,5,6,7,9,11,12,13) \qquad [4.25]$$

the steps to determine the essential prime implicants are given in Table 4.2. The addition operations required to obtain the different terms are as follows:

$$t_1 = m_2 + m_3 \quad t_2 = m_2 + m_6 \quad t_3 = m_4 + m_5 \quad t_4 = m_4 + m_6$$
$$t_5 = m_4 + m_{12} \quad t_6 = m_3 + m_7 \quad t_7 = m_3 + m_{11} \quad t_8 = m_5 + m_7$$
$$t_9 = m_5 + m_{13} \quad t_{10} = m_6 + m_7 \quad t_{11} = m_9 + m_{11} \quad t_{12} = m_9 + m_{13}$$
$$t_{13} = m_{12} + m_{13} \qquad t_{14} = t_1 + t_{10} = t_2 + t_6$$
$$t_{15} = t_3 + t_{10} = t_4 + t_8 \qquad t_{16} = t_3 + t_{13} = t_5 + t_9$$

	Column 1	Column 2	Column 3
Group 1	m_2 : 0010 ✓ m_4 : 0100 ✓	t_1 : 001- ✓ t_2 : 0-10 ✓ t_3 : 010- ✓ t_4 : 01-0 ✓ t_5 : -100 ✓	
Group 2	m_3 : 0011 ✓ m_5 : 0101 ✓ m_6 : 0110 ✓ m_9 : 1001 ✓ m_{12} : 1100 ✓	t_6 : 0-11 ✓ t_7 : -011 ⋆ t_8 : 01-1 ✓ t_9 : -101 ✓ t_{10} : 011- ✓ t_{11} : 10-1 ⋆ t_{12} : 1-01 ⋆ t_{13} : 110- ✓	t_{14} : 0-1- ⋆ t_{15} : 01- - ⋆ t_{16} : -10- ⋆
Group 3	m_7 : 0111 ✓ m_{11} : 1011 ✓ m_{13} : 1101 ✓		

Table 4.2. *Table to determine prime implicants*

4.4.1.2. *Iterated consensus method*

The iterated consensus method is another technique that can be used to determine the prime implicants of a logic function. It makes use of the consensus of terms, which translates into the logic expressions: $X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z$ and the absorption law, $X + XY = X$, to suppress the redundant terms.

For two logic expressions, we can have a consensus term, null consensus or no consensus.

EXAMPLE 4.6.– Determine the nature of consensus for the following expressions:

– $A \cdot C$ and $B \cdot \overline{D}$ have no consensus;

– $A \cdot B \cdot \overline{C}$ and $\overline{A} \cdot C \cdot \overline{D}$ have null consensus;

– $A \cdot B \cdot C$ and $\overline{A} \cdot D$ have $B \cdot C \cdot D$ exhibit a consensus term.

The consensus theorem can be applied to any sum-of-products form (canonical or reduced) of a logic function.

To determine the prime implicants of a logic function, a list of its terms is first established. The possible consensus are then singled out by considering the terms two by two.

There is a consensus between two terms, t_p and t_q, with respect to one of the variables if this variable appears in one of the terms while its complement appears in the other term. The consensus term, $C(t_q, t_p)$, is then equal to the product of the factors of t_p and t_q other than this variable and its complement.

A consensus term is added to the list of terms if it is not identical to any term that already appears on the list or it is not included in a term that is already on the list.

With the addition of each consensus term we try to reduce the number of terms in the list by eliminating all terms that are included in other terms.

This process is repeated till no new consensus can be obtained. The terms that remain on the list are the prime implicants.

Determine the prime implicants of the following logic function:

$$F(A, B, C, D) = \sum m(2, 3, 4, 5, 6, 7, 9, 11, 12, 13) \qquad [4.26]$$

Using the property $X \cdot Y + X \cdot \overline{Y} = X$ to group the terms, we obtain:

$$\begin{aligned} F(A,B,C,D) &= \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} \\ &+ \overline{A} \cdot B \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot D \\ &+ A \cdot \overline{B} \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot C \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D} \\ &+ A \cdot B \cdot \overline{C} \cdot D \qquad [4.27] \\ &= \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B + A \cdot \overline{B} \cdot D + A \cdot B \cdot \overline{C} \qquad [4.28] \end{aligned}$$

We can, thus, choose to begin looking for prime implicants with the terms $\overline{A} \cdot \overline{B} \cdot C$, $\overline{A} \cdot B$, $A \cdot \overline{B} \cdot D$ and $A \cdot B \cdot \overline{C}$. All the consensuses used are given in Table 4.3.

~~$t_1 : \overline{A} \cdot \overline{B} \cdot C$~~	
$t_2 : \overline{A} \cdot B$	$C(t_2, t_1)$ (Add t_5, suppress t_1)
$t_3 : A \cdot \overline{B} \cdot D$	$C(t_3, t_2)$
~~$t_4 : A \cdot B \cdot \overline{C}$~~	$C(t_4, t_2)$ (Add t_6, suppress t_4)
$t_5 : \overline{A} \cdot C$	$C(t_5, t_2)$; $C(t_5, t_3)$ (Add t_7)
$t_6 : B \cdot \overline{C}$	$C(t_6, t_2)$; $C(t_6, t_3)$ (Add t_8); $C(t_6, t_5) = t_2$
$t_7 : \overline{B} \cdot C \cdot D$	$C(t_7, t_2) \subset t_5$; $C(t_7, t_3)$; $C(t_7, t_5)$; $C(t_7, t_6)$
$t_8 : A \cdot \overline{C} \cdot D$	$C(t_8, t_2) \subset t_6$; $C(t_8, t_3)$; $C(t_8, t_5)$; $C(t_8, t_6)$; $C(t_8, t_7) = t_3$

Table 4.3. *Table to determine the prime implicants using the consensus theorem*

To simplify the iterated consensus method, we can adopt the digital form of representing terms in the logic function to be simplified. A complemented variable is represented by 0, a non-complemented variable is represented by 1 and a missing variable is represented by a hyphen (-). Consensus can exist between two terms when a variable is represented by 0 in one term and 1 in the other. In the consensus term, a given variable is represented by 1 if it is represented by 1 in one of the terms and either 1 or - in the other; it is represented by 0 if it is represented by 0 in one of the terms and either 0 or - in the other; and it is represented by - if it is represented by 0 in one term and 1 in the other or if it is represented by - in both terms.

4.4.2. *Finding the constitutive terms of a minimal expression*

Finding the minimum number of prime implicants that represents a logic function is often formulated as a minimal cover problem. For a single output logic function, this can lead to a two-input table. The prime implicants are entered in the rows and the minterms for the logic function to be simplified are entered in the columns. A cross is placed at the intersection of a row and a column to indicate that a minterm is covered by a prime implicant.

The objective is to cover all the minterms of the logic function by using a minimum set of prime implicants. To achieve this, first a single cross serves as the identifying mark of the essential prime implicants, which must necessarily be a part of the minimal expression and which are associated with each column. Each essential prime implicant represents a single choice for covering a given minterm. Once the essential prime implicants are determined, it only remains to choose, from the other prime implicants, those that allow for the implementation of a minimal cover of the remaining minterms.

4.4.2.1. *Graphical method for the reduction of a prime implicant chart*

The dominance relation between prime implicants and the dominance relations between minterms can be used to reduce the prime implicant chart.

It is necessary to introduce some definitions before implementing the graphical methods for the reduction of a prime implicant chart:

– two identical rows (columns) are said to be interchangeable;

– let i and j be two rows in a prime implicant chart. The row i can be said to dominate j if the essential prime implicant associated with i covers at least one minterm more than the minterms covered by the essential prime implicants associated with j.

When the row i dominates j, there exists a minimized sum-of-products form that does not include the essential prime implicant associated with j. The dominated line can, thus, be suppressed;

– let k and l be two columns in a prime implicant chart. The column k is said to dominate l if the minterm associated with k is covered by at least one essential prime implicant more than the essential prime implicants covering the minterm associated with l.

When the column k dominates l, an essential implicant covering the minterm associated with l also covers the minterm associated with k. We can, thus, suppress the dominant column.

The implementation of the graphical method for the reduction of a prime implicant chart is carried out as follows:

1) find the essential prime implicants and eliminate them;

2) identify the dominance relations between the prime implicants (rows) and eliminate the dominated rows;

3) identify the dominance relations between the minterms (columns) and eliminate the dominant columns.

Repeat steps 1, 2 and 3, until there are no more possibilities for elimination: if there are no more rows and columns, a minimal solution has been found; if not, the prime implicant chart is said to be cyclic.

NOTE 4.3.– The reduction of the prime implicant chart following the rules given above is nothing but the graphical translation of the absorption theorems. The solution obtained depends on the choice of the dominance relations. The graphical method for the reduction of a prime implicant chart can only be easily applied to cases where the possible choices for the dominance relations are limited.

EXAMPLE 4.7.– Minimize the following logic function:

$$F(A,B,C,D) = \sum m(2,3,4,5,6,7,9,11,12,13) \qquad [4.29]$$

The prime implicant chart in Table 4.4 can be constructed by applying the graphical method to the minimization of the function F. Here, a cross within a circle is placed at the intersection of a row and a column associated with a single prime implicant. We have two essential prime implicants that cover all the minterms except for $A \cdot \overline{B} \cdot \overline{C} \cdot D$ and $A \cdot \overline{B} \cdot C \cdot D$. To cover these minterms, we choose $A \cdot \overline{B} \cdot D$, instead of the prime implicants $A \cdot \overline{C} \cdot D$ and $\overline{B} \cdot C \cdot D$. As a result, the single minimized form of F can be obtained as follows:

$$F(A,B,C,D) = \overline{A} \cdot C + B \cdot \overline{C} + A \cdot \overline{B} \cdot D \qquad [4.30]$$

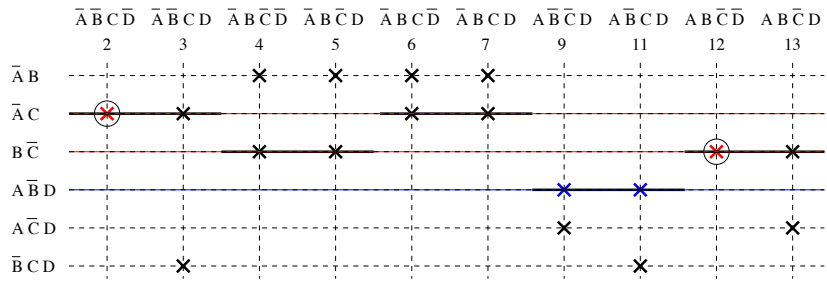


Table 4.4. *Prime implicant chart for F*

When a logic function has more than one minimized form, the choice of the non-essential prime implicants may not be evident.

NOTE 4.4.– There are logic functions that do not have essential prime implicants. Each column of the prime implicant chart for such a function contains at least two crosses. The prime implicant chart is then said to be cyclic.

In the case of the logic functions [4.31] and [4.34], using the Karnaugh maps shown in Figures 4.17 and 4.18 we obtain two minimized sum-of-products forms as follows:

$$F(A,B,C,D) = \sum m(0,4,6,8,10,14) \quad [4.31]$$

$$= A \cdot \overline{B} \cdot \overline{D} + \overline{A} \cdot \overline{C} \cdot \overline{D} + B \cdot C \cdot \overline{D} \quad [4.32]$$

$$= \overline{A} \cdot B \cdot \overline{D} + A \cdot C \cdot \overline{D} + \overline{B} \cdot \overline{C} \cdot \overline{D} \quad [4.33]$$

and:

$$F(A,B,C,D) = \sum m(0,1,4,5,6,7,8,9,10,11,14,15) \quad [4.34]$$

$$= A \cdot \overline{B} + \overline{A} \cdot \overline{C} + B \cdot C \quad [4.35]$$

$$= \overline{A} \cdot B + A \cdot C + \overline{B} \cdot \overline{C} \quad [4.36]$$

We have six prime implicants for each of these functions but no essential prime implicant.

In general, it is always possible to find a function of n variables, whose Karnaugh map includes the cells encompassing 2^{n-1} minterms and which has a cyclic prime implicant chart.

4.4.2.2. *Petrick's method*

Petrick's method is a technique that can be used to determine, in a systematic manner, all the minimal sum-of-products form from the prime implicant chart of a logic function. It is especially useful when there are several solutions to be determined and the number of prime implicants is high.

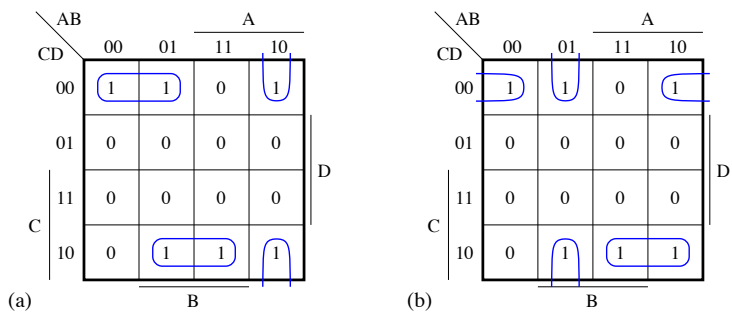


Figure 4.17. *Karnaugh map for the logic function* $F(A,B,C,D) = \sum m(0,4,6,8,10,14)$

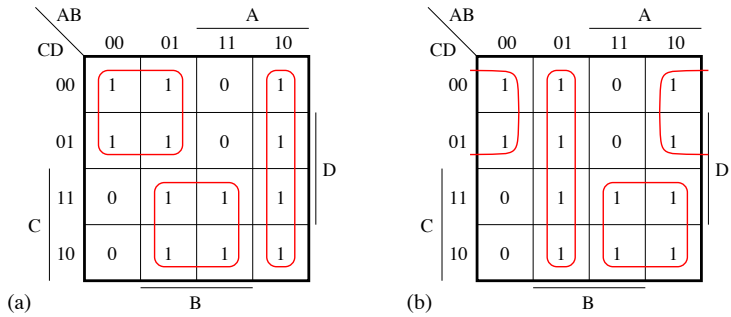


Figure 4.18. *Karnaugh map for the logic function* $F(A,B,C,D) = \sum m(0,1,4,5,6,7,8,9,10,11,14,15)$

The implementation of Petrick's method uses the following steps:

1) reduce the prime implicant chart by eliminating the rows and columns associated with essential prime implicants;

2) attribute the denominations P_1, P_2, P_3, $\cdots$, to different rows in the reduced prime implicant chart;

3) form a logic function P that is true when all the columns are covered. The function P is equal to a product of sums, where each sum is in the form $(P_{i0} + P_{i1} + \cdots)$, with P_{i0}, P_{i1} being related to the rows that cover the column i;

4) express P as a minimum sum of products by expanding and reducing the terms of the multiplication using the logic identity $X + XY = X$;

5) each term of the result represents a solution, that is, a set of rows that covers all the minterms of the chart. To determine the solutions with lowest hardware implementation costs, it is necessary to find terms that contain a minimum number of variables. Each of these terms represents a solution having a minimum number of prime implicants;

6) for each of the terms obtained in the previous step, count the number of variables that form each prime implicant and determine the total number of variables. Choose the term or terms that have the minimum total number of variables and write the corresponding sum of prime implicants.

EXAMPLE 4.8.– Determine all the minimized sum-of-products forms for the following logic function:

$$F(A,B,C,D) = \sum m(0,2,6,7,8,9,10,13,15) \qquad [4.37]$$

Table 4.5 gives the following seven prime implicants:

$$\overline{B} \cdot \overline{D},\ A \cdot B \cdot D,\ B \cdot C \cdot D,\ A \cdot \overline{C} \cdot D,\ \overline{A} \cdot B \cdot C,\ A \cdot \overline{B} \cdot \overline{C} \text{ and } \overline{A} \cdot C \cdot \overline{D}.$$

	Column 1	Column 2	Column 3
Group 0	m_0 : 0000 ✓	t_1 : 00-0 ✓ t_2 : -000 ✓	t_{11} : -0-0 ⋆
Group 1	m_2 : 0010 ✓ m_8 : 1000 ✓	t_3 : 0-10 ⋆ t_4 : -010 ✓ t_5 : 100- ⋆ t_6 : 10-0 ✓	
Group 2	m_6 : 0110 ✓ m_9 : 1001 ✓ m_{10} : 1010 ✓	t_7 : 011- ⋆ t_8 : 1-01 ⋆	
Group 3	m_7 : 0111 ✓ m_{13} : 1101 ✓	t_9 : -111 ⋆ t_{10} : 11-1 ⋆	
Group 4	m_{15} : 1111 ✓		

Table 4.5. *Table for the determination of prime implicants*

Table 4.6 gives the complete and reduced form of the prime implicant chart for the logic function F. As can be observed in the complete Table 4.6(a), $\overline{B} \cdot \overline{D}$ is the only essential prime implicant. The reduced form of the prime implicant chart shown in Table 4.6(b) is useful for the implementation of Petrick's method. To cover all the minterms of F, the following logic function must be true:

$$P = (P_1 + P_3)(P_3 + P_5)(P_2 + P_4)(P_4 + P_6)(P_5 + P_6) \qquad [4.38]$$

Using the logic identity $(X + Y)(X + Z) = X + Y \cdot Z$, we can write:

$$P = (P_3 + P_1 \cdot P_5)(P_4 + P_2 \cdot P_6)(P_5 + P_6) \qquad [4.39]$$

Elaborating the function P, we obtain:

$$P = P_1 \cdot P_4 \cdot P_5 + P_2 \cdot P_3 \cdot P_6 + P_3 \cdot P_4 \cdot P_5 + P_3 \cdot P_4 \cdot P_6 + P_1 \cdot P_2 \cdot P_5 \cdot P_6 + P_1 \cdot P_4 \cdot P_5 \cdot P_6 + P_2 \cdot P_3 \cdot P_5 \cdot P_6 \qquad [4.40]$$

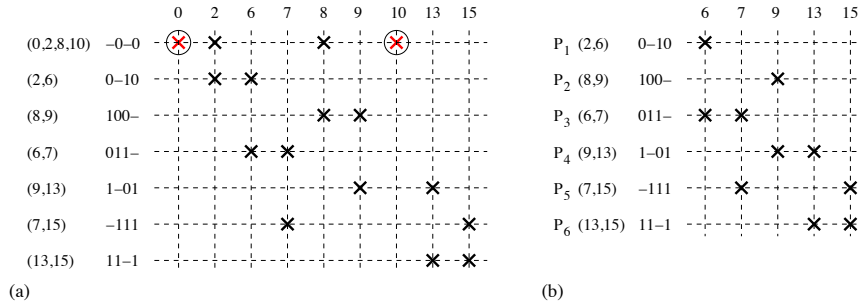


Table 4.6. *Prime implicant chart for F: a) complete form and b) reduced form*

Because $X + X \cdot Y = X$, we have:

$$P = P_1 \cdot P_4 \cdot P_5 + P_2 \cdot P_3 \cdot P_6 + P_3 \cdot P_4 \cdot P_5 + P_3 \cdot P_4 \cdot P_6 + P_1 \cdot P_2 \cdot P_5 \cdot P_6 \quad [4.41]$$

For the function P to be true, it is sufficient that one of its terms be true. There are, thus, five possible solutions but we can only retain those that exhibit a minimum number of rows, namely, P_1 and P_4 and P_5, or P_2 and P_3 and P_6, or P_3 and P_4 and P_5, or P_3 and P_4 and P_6. To express the minimal forms of F, we must combine these solutions with the only essential prime implicant.

The function F, therefore, has four minimal forms, which are:

$$F(A,B,C,D) = \overline{B} \cdot \overline{D} + \overline{A} \cdot C \cdot \overline{D} + B \cdot C \cdot D + A \cdot \overline{C} \cdot D \quad [4.42]$$

$$= \overline{B} \cdot \overline{D} + \overline{A} \cdot B \cdot C + B \cdot C \cdot D + A \cdot \overline{C} \cdot D \quad [4.43]$$

$$= \overline{B} \cdot \overline{D} + \overline{A} \cdot B \cdot C + A \cdot B \cdot D + A \cdot \overline{C} \cdot D \quad [4.44]$$

$$= \overline{B} \cdot \overline{D} + \overline{A} \cdot B \cdot C + A \cdot B \cdot D + A \cdot \overline{B} \cdot \overline{C} \quad [4.45]$$

EXAMPLE 4.9.– Find all the minimized sum-of-products forms of the following logic function:

$$F(A,B,C,D) = \sum m(0,2,3,4,5,6,7,8,9,10,11,12,13) \quad [4.46]$$

The list of prime implicants can be deduced from Table 4.7 as follows:

$\overline{A} \cdot \overline{D}$, $\overline{B} \cdot \overline{D}$, $\overline{C} \cdot \overline{D}$, $\overline{A} \cdot C$, $\overline{B} \cdot C$, $\overline{A} \cdot B$, $B \cdot \overline{C}$, $A \cdot \overline{B}$ and $A \cdot \overline{C}$.

	Column 1	Column 2	Column 3
Group 0	m_0 : 0000 ✓	t_1 : 00-0 ✓	t_{22} : 0- -0 ⋆
		t_2 : 0-00 ✓	t_{23} : -0-0 ⋆
		t_3 : -000 ✓	t_{24} : - -00 ⋆
Group 1	m_2 : 0010 ✓	t_4 : 001- ✓	t_{25} : 0-1- ⋆
	m_4 : 0100 ✓	t_5 : 0-10 ✓	t_{26} : -01- ⋆
	m_8 : 1000 ✓	t_6 : -010 ✓	t_{27} : 01- - ⋆
		t_7 : 010- ✓	t_{28} : -10- ⋆
		t_8 : 01-0 ✓	t_{29} : 10- - ⋆
		t_9 : -100 ✓	t_{30} : 1-0- ⋆
		t_{10} : 100- ✓	
		t_{11} : 10-0 ✓	
		t_{12} : 1-00 ✓	
Group 2	m_3 : 0011 ✓	t_{13} : 0-11 ✓	
	m_5 : 0101 ✓	t_{14} : -011 ✓	
	m_6 : 0110 ✓	t_{15} : 01-1 ✓	
	m_9 : 1001 ✓	t_{16} : -101 ✓	
	m_{10} : 1010 ✓	t_{17} : 011- ✓	
	m_{12} : 1100 ✓	t_{18} : 10-1 ✓	
		t_{19} : 1-01 ✓	
		t_{20} : 101- ✓	
		t_{21} : 110- ✓	
Group 3	m_7 : 0111 ✓		
	m_{11} : 1011 ✓		
	m_{13} : 1101 ✓		

Table 4.7. *Table for the determination of prime implicants*

The list of prime implicants can be deduced from Table 4.7 as follows:

$$\overline{A} \cdot \overline{D},\ \overline{B} \cdot \overline{D},\ \overline{C} \cdot \overline{D},\ \overline{A} \cdot C,\ \overline{B} \cdot C,\ \overline{A} \cdot B,\ B \cdot \overline{C},\ A \cdot \overline{B} \text{ and } \quad A \cdot \overline{C}.$$

Table 4.8 gives the prime implicant chart for the function F. We can observe that each minterm is covered by at least two terms and, consequently, there is no essential prime implicant.

As dominance relations can be established between the essential implicants or between the minterms, the following step consists of suppressing the rows associated with dominated essential implicants and columns corresponding to dominant minterms. Thus, column 2 dominates 3, 4 dominates 5, 6 dominates 7, 8 dominates 9, 10 dominates 11 and 12 dominates 13. As a result, columns 2, 4, 6, 8, 10 and 12 can be eliminated as illustrated in Table 4.9(a). Essential implicants, 0- -0, -0-0 and - -00,

are interchangeable. We can, thus, eliminate two of them. Choosing to retain the term 0- -0, the prime implicant chart reduces to that in Table 4.9(b) and the term 0- -0 becomes a secondary essential prime implicant, the elimination of which leads to the construction in Table 4.9(c).

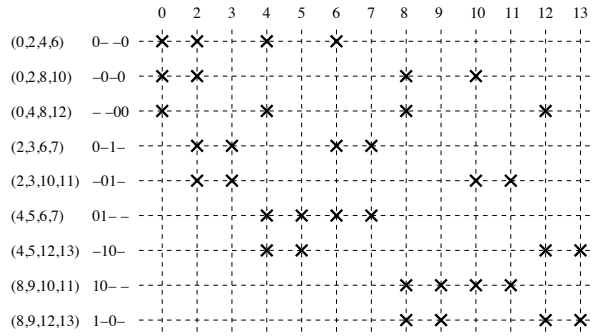
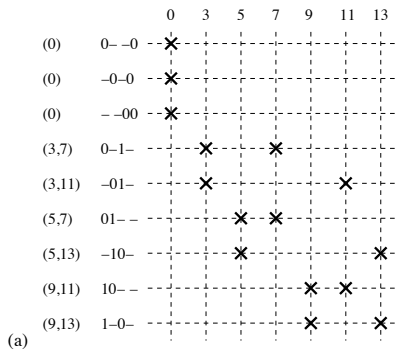


Table 4.8. *Prime implicant chart for F*



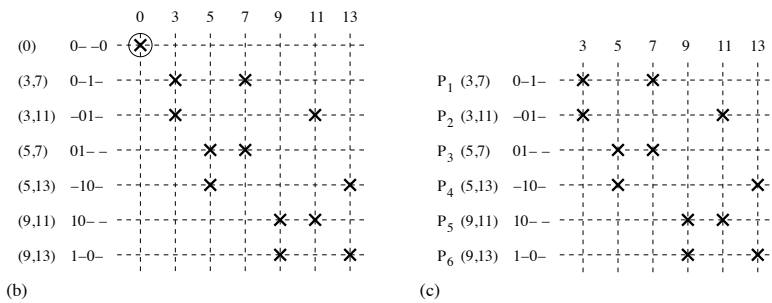


Table 4.9. *Reduced forms of the prime implicant charts for F*

Using Petrick's methods, the overlap equation can be put into the following form:

$$P = (P_1 + P_2)(P_3 + P_4)(P_1 + P_3)(P_5 + P_6)(P_2 + P_5)(P_4 + P_6) \quad [4.47]$$

Expanding P, taking into account the logic identity:

$$(X + Y)(X + Z) = X + Y \cdot Z$$

we have:

$$P = P_1P_4P_5 + P_1P_2P_3P_6 + P_1P_3P_5P_6 + P_1P_2P_4P_6 + P_2P_3P_6 + P_2P_3P_4P_5 + P_2P_3P_5P_6 + P_2P_3P_4P_6 \quad [4.48]$$

Each term in the sum is a possible solution and we thus have eight possible solutions. However, only the two simplest terms, $P_1P_4P_5$ and $P_2P_3P_6$, can be retained to implement the minimal forms of F. The term $P_1P_4P_5$ is made up of $\overline{A} \cdot C$, $B \cdot \overline{C}$ and $A \cdot \overline{B}$, while $P_2P_3P_6$ consists of $\overline{B} \cdot C$, $\overline{A} \cdot B$ and $A \cdot \overline{C}$. Finally, we can consider the following minimal forms:

$$F = \overline{A} \cdot \overline{D} + \overline{A} \cdot C + B \cdot \overline{C} + A \cdot \overline{B} \quad [4.49]$$

$$= \overline{A} \cdot \overline{D} + \overline{B} \cdot C + \overline{A} \cdot B + A \cdot \overline{C} \quad [4.50]$$

Choosing either -0-0 or - -00, instead of 0- -0, we obtain:

$$F = \overline{B} \cdot \overline{D} + \overline{A} \cdot C + B \cdot \overline{C} + A \cdot \overline{B} \quad [4.51]$$

$$= \overline{B} \cdot \overline{D} + \overline{B} \cdot C + \overline{A} \cdot B + A \cdot \overline{C} \quad [4.52]$$

or:

$$F = \overline{C} \cdot \overline{D} + \overline{A} \cdot C + B \cdot \overline{C} + A \cdot \overline{B} \quad [4.53]$$

$$= \overline{C} \cdot \overline{D} + \overline{B} \cdot C + \overline{A} \cdot B + A \cdot \overline{C} \quad [4.54]$$

There are, thus, six minimal forms for the logic function F.

EXAMPLE 4.10.– Determine all the minimized sum-of-products forms for the following logic function:

$$F(A, B, C, D, E) = \sum m(5, 7, 8, 9, 10, 11, 13, 15, 21, 23, 26, 28, 29, 30, 31) \quad [4.55]$$

The prime implicants of the logic function F are:

-1010, 11-10, 010- -, 01- -1, 111- - and - -1-1

or:

$$B \cdot \overline{C} \cdot D \cdot \overline{E}, \quad A \cdot B \cdot D \cdot \overline{E}, \quad \overline{A} \cdot B \cdot \overline{C}, \quad \overline{A} \cdot B \cdot E, \quad A \cdot B \cdot C \quad \text{and} \quad C \cdot E.$$

	Column 1	Column 2	Column 3	Column 4
Group 1	m_8 : 01000 ✓	t_1 : 0100- ✓ t_2 : 010-0 ✓	t_{24} : 010-- ⋆	
Group 2	m_5 : 00101 ✓ m_9 : 01001 ✓ m_{10} : 01010 ✓	t_3 : 001-1 ✓ t_4 : 0-101 ✓ t_5 : -0101 ✓ t_6 : 010-1 ✓ t_7 : 01-01 ✓ t_8 : 0101- ✓ t_9 : -1010 ⋆	t_{25} : 0-1-1 ✓ t_{26} : -01-1 ✓ t_{27} : --101 ✓ t_{28} : 01--1 ⋆	t_{33} : ---1-1 ⋆
Group 3	m_7 : 00111 ✓ m_{11} : 01011 ✓ m_{13} : 01101 ✓ m_{21} : 10101 ✓ m_{26} : 11010 ✓ m_{28} : 11100 ✓	t_{10} : 0-111 ✓ t_{11} : -0111 ✓ t_{12} : 01-11 ✓ t_{13} : 011-1 ✓ t_{14} : -1101 ✓ t_{15} : 101-1 ✓ t_{16} : 1-101 ✓ t_{17} : 11-10 ⋆ t_{18} : 1110- ✓ t_{19} : 111-0 ✓	t_{29} : ---111 ✓ t_{30} : -11-1 ✓ t_{31} : 1-1-1 ✓ t_{32} : 111-- ⋆	
Group 4	m_{15} : 01111 ✓ m_{23} : 10111 ✓ m_{29} : 11101 ✓ m_{30} : 11110 ✓	t_{20} : -1111 ✓ t_{21} : 1-111 ✓ t_{22} : 111-1 ✓ t_{23} : 1111- ✓		
Group 5	m_{31} : 11111 ✓			

Table 4.10. *Table for the determination of prime implicants*

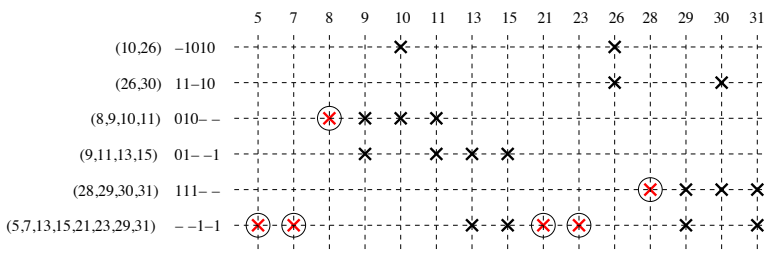


Table 4.11. *Prime implicant chart for F*

The prime implicant chart for the logic function $F(A, B, C, D, E)$ is represented in Table 4.11. The minterm m_8 is only covered by the term 010- -; the minterm m_{28} is only covered by 111- -; and the minterms m_5, m_7, m_{21}, and m_{23} are covered only by - -1-1. Thus, the terms 010- - ($\overline{A} \cdot B \cdot \overline{C}$), 111- - ($A \cdot B \cdot C$) and - -1-1 ($C \cdot E$) are the essential prime implicants. As the essential prime implicants are part of any minimized form of a logic function, it only remains to cover the minterm m_{26}. This can be done by using either -1010 ($B \cdot \overline{C} \cdot D \cdot \overline{E}$) or 11-10 ($A \cdot B \cdot D \cdot \overline{E}$). We thus obtain the following two solutions:

$$F = A \cdot B \cdot C + C \cdot E + \overline{A} \cdot B \cdot \overline{C} + B \cdot \overline{C} \cdot D \cdot \overline{E} \quad [4.56]$$

$$= A \cdot B \cdot C + C \cdot E + \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot D \cdot \overline{E} \quad [4.57]$$

4.4.3. *Quine–McCluskey technique: simplification of incompletely defined functions*

Consider the following incompletely defined logic function:

$$F(A, B, C, D) = \sum m(1, 7, 9, 10, 11, 13) + \sum d(5, 8, 15) \quad [4.58]$$

To determine the prime implicants for such a function, we construct a table, where the incompletely defined terms are treated in the same way as minterms.

Table 4.12 gives the following prime implicants: $A \cdot \overline{B}$, $A \cdot D$, $B \cdot D$ and $\overline{C} \cdot D$.

In the prime implicants chart in Table 4.13, we insert only the minterms and not the incompletely defined terms. It can be deduced from the chart that $A \cdot \overline{B}$, $B \cdot D$ and $\overline{C} \cdot D$ are the essential prime implicants of the function F. As all the minterms for the function F are covered by the essential prime implicants, the minimized sum-of-products can be written as follows:

$$F(A, B, C, D) = A \cdot \overline{B} + B \cdot D + \overline{C} \cdot D \quad [4.59]$$

In the case of the simplification of an incompletely defined function using the Quine–McCluskey method, the incompletely defined terms are taken into account only for the process of determining the prime implicants and are not inserted in the prime implicants chart.

4.4.4. *Simplification of functions with multiple outputs*

In general, the simplification of logic functions with multiple outputs consists of minimizing several functions simultaneously. The hardware implementation cost

function to be minimized has a bearing on several logic functions at the same time and it is thus necessary to take into account the interdependence between the different logic functions.

	Column 1	Column 2	Column 3
Group 1	m_1 : 0001 ✓ m_8 : 1000 ✓	t_1 : 0-01 ✓ t_2 : -001 ✓ t_3 : 100- ✓ t_4 : 10-0 ✓	t_{13} : - -01 ⋆ t_{14} : 10- - ⋆
Group 2	m_5 : 0101 ✓ m_9 : 1001 ✓ m_{10} : 1010 ✓	t_5 : 01-1 ✓ t_6 : -101 ✓ t_7 : 10-1 ✓ t_8 : 1-01 ✓ t_9 : 101- ✓	t_{15} : -1-1 ⋆ t_{16} : 1- -1 ⋆
Group 3	m_7 : 0111 ✓ m_{11} : 1011 ✓ m_{13} : 1101 ✓	t_{10} : -111 ✓ t_{11} : 1-11 ✓ t_{12} : 11-1 ✓	
Group 4	m_{15} : 1111 ✓		

Table 4.12. *Table for the determination of prime implicants*

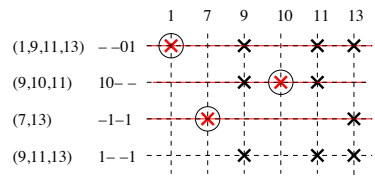


Table 4.13. *Prime implicants chart for F*

To establish the list of prime implicants in the case of functions with multiple outputs, it is necessary to modify the Quine–McCluskey method or the iterated consensus method to take into account the tag associated with each term. The number of tag bits correspond to the number of functions. The logic state 1 (or 0) of a bit can be used to identify whether a term belongs (or does not belong) to a given function. The tag associated with the term resulting from the logic combination of two other terms is obtained by multiplying their tags, bit by bit.

Let us consider the following incompletely defined functions:

$$F(A,B,C,D) = \sum m(1,3,4,10,11,12,14) + \sum d(6,7,8,9) \quad [4.60]$$

$$G(A,B,C,D) = \sum m(1,2,4,10,14) + \sum d(5,6,9,13) \quad [4.61]$$

Assuming that these two logic functions characterize a 4-input and 2-output system, determine their minimized sum-of-products equations.

The two methods used most often to determine the prime implicants of a logic function are the following: the Quine–McCluskey method and the iterated consensus algorithm.

4.4.4.1. *Prime implicant determination using the Quine–McCluskey method*

We construct a table in which we insert the minterms and the incompletely defined terms grouped according to the number of 1s in their representation and a tag indicating the function for which a term can be used.

Going through Table 4.14, we note that there are four prime implicants common to both functions F and G, seven prime implicants associated only with F and three prime implicants associated only with G.

4.4.4.2. *Prime implicant determination using the iterated consensus method*

To apply the consensus method, it is necessary to draw up a list of minterms for each function as well as all the possible products of functions. A tag is then associated with each minterm. The determination of the consensus term for each pair of minterms makes it possible to either add a new term or suppress the terms that are included in other terms. In the specific case of functions with multiple outputs, we also suppress any new term whose tag is formed only of bits set to 0. The prime implicants are those terms that remain in the table at the end of the process.

In Table 4.15, we have four prime implicants common to both functions F and G, seven prime implicants associated only with F and three prime implicants associated only with G.

4.4.4.3. *Prime implicant chart*

In general, in the case of simplification of functions with multiple outputs, the prime implicant chart includes a section with prime implicants common to all functions and a section for each group of prime implicants that only belong to a single function.

Table 4.16 gives the prime implicants chart for the functions F, G and $F \cdot G$. It should be noted that the row associated with the term 011- is not selected for any

function. This is explained by the fact that the essential prime implicant 011- belongs uniquely to the incompletely defined terms. The function F has an essential prime implicant, -0-1 ($\overline{B} \cdot D$), exactly like the function G, whose essential prime implicant is –10 ($C \cdot \overline{D}$). Eliminating the rows associated with these essential prime implicants, as well as the columns covered by them, we obtain the reduced form of the prime implicants chart represented in Table 4.17. To complete the overlap of the logic functions, we choose the prime implicant 1- -0 ($A \cdot \overline{D}$), which covers three minterms (10,12,14) of F, the prime implicant 01-0 ($\overline{A} \cdot B \cdot \overline{D}$), which covers the minterm (4) of F and G, and the prime implicant - -01 ($\overline{C} \cdot D$), which covers the minterm (1) of G. The minimal forms of F and G are, thus, written as:

$$F(A, B, C, D) = A \cdot \overline{D} + \overline{B} \cdot D + \overline{A} \cdot B \cdot \overline{D} \qquad [4.62]$$

	Column 1 F G	Column 2 F G	Column 3 F G
Group 1	m_1 : 0001 1 1 ✓ m_2 : 0010 0 1 ✓ m_4 : 0100 1 1 ✓ m_8 : 1000 1 0 ✓	t_1 : 00-1 1 0 ✓ t_2 : 0-01 0 1 ✓ t_3 : -001 1 1 ⋆ t_4 : 0-10 0 1 ✓ t_5 : -010 0 1 ✓ t_6 : 010- 0 1 ⋆ t_7 : 01-0 1 1 ⋆ t_8 : -100 1 0 ✓ t_9 : 100- 1 0 ✓ t_{10} : 10-0 1 0 ✓ t_{11} : 1-00 1 0 ✓	t_{21} : -0-1 1 0 ⋆ t_{22} : - -01 0 1 ⋆ t_{23} : -0-1 1 0 ⋆ t_{24} : - -10 0 1 ⋆ t_{25} : -1-0 1 0 ⋆ t_{26} : 10- - 1 0 ⋆ t_{27} : 1- -0 1 0 ⋆
Group 2	m_3 : 0011 1 0 ✓ m_5 : 0101 0 1 ✓ m_6 : 0110 1 1 ✓ m_9 : 1001 1 1 ✓ m_{10} : 1010 1 1 ✓ m_{12} : 1100 1 0 ✓	t_{12} : 0-11 1 0 ⋆ t_{13} : -011 1 0 ✓ t_{14} : -101 0 1 ✓ t_{15} : 011- 1 0 ⋆ t_{16} : -110 1 1 ⋆ t_{17} : 10-1 1 0 ✓ t_{18} : 1-01 0 1 ✓ t_{19} : 101- 10 ✓ t_{20} : 1-10 11 ⋆ t_{21} : 11-0 10 ✓	
Group 3	m_7 : 0111 1 0 ✓ m_{11} : 1011 1 0 ✓ m_{13} : 1101 0 1 ✓ m_{14} : 1110 1 1 ✓		

Table 4.14. *Table for the determination of prime implicants*

	F	G	
~~m_1 : 0001~~	1	1	
~~m_2 : 0010~~	0	1	
~~m_3 : 0011~~	1	0	$C(m_3, m_1)$ (Add t_1, Suppress m_3)
~~m_4 : 0100~~	1	1	$C(m_5, m_1)$ (Add t_2, Suppress m_5)
~~m_5 : 0101~~	0	1	$C(m_9, m_1)$ (Add t_3, Suppress m_9 and m_1)
~~m_6 : 0110~~	1	1	$C(m_6, m_2)$ (Add t_4, Suppress m_2)
~~m_7 : 0111~~	1	0	$C(m_6, m_4)$ (Add t_5, Suppress m_6 and m_4)
~~m_8 : 1000~~	1	0	$C(m_{10}, m_8)$ (Add t_6, Suppress m_8)
~~m_9 : 1001~~	1	1	$C(m_{11}, m_{10})$ (Add t_7, Suppress m_{11})
~~m_{10} : 1010~~	1	1	$C(m_{14}, m_{10})$ (Add t_8, Suppress m_{14} and m_{10})
~~m_{11} : 1011~~	1	0	$C(t_1, m_7)$ (Add t_9, Suppress m_7)
~~m_{12} : 1100~~	1	0	$C(t_5, m_{12})$ (Add t_{10}, Suppress m_{12})
~~m_{13} : 1101~~	0	1	$C(t_2, m_{13})$ (Add t_{11}, Suppress m_{13})
~~m_{14} : 1110~~	1	1	$C(t_7, t_1)$ (Add t_{12})
~~t_1 : -00-1~~	1	0	$C(t_5, t_2)$ (Add t_{13})
~~t_2 : 0-01~~	0	1	$C(t_6, t_3)$ (Add t_{14})
t_3 : -001	1	1	$C(t_7, t_3)$ (Add t_{15})
~~t_4 : 0-10~~	0	1	$C(t_9, t_3)$ (Add t_{16})
t_5 : 01-0	1	1	$C(t_{11}, t_3)$ (Add t_{17}, Suppress t_{11}, t_2)
~~t_6 : 10-0~~	1	0	$C(t_{12}, t_3)$ (Add t_{18}, Suppress t_{16}, t_{12}, t_1)
			$C(t_{13}, t_3) \subset t_{17}$
~~t_7 : 101-~~	1	0	$C(t_8, t_4)$ (Add t_{19}, Suppress t_4)
t_8 : 1-10	1	1	$C(t_8, t_5)$ (Add t_{20})
t_9 : 0-11	1	0	$C(t_9, t_5)$ (Add t_{21})
			$C(t_{17}, t_5) = t_{13}$
~~t_{10} : -100~~	1	0	$C(t_{10}, t_6)$ (Add t_{22})
~~t_{11} : -101~~	0	1	$C(t_{15}, t_6)$ (Add t_{23}, Suppress t_{15}, t_{14}, t_7, t_6)
			$C(t_9, t_7) \subset t_{18}; C(t_{14}, t_7) = t_{23}; C(t_{16}, t_7) \subset t_{18}$
			$C(t_{22}, t_7) = C(t_{14}, t_8) = C(t_{15}, t_8) = C(t_{18}, t_8) \subset t_{23}$
~~t_{12} : -011~~	1	0	$C(t_{22}, t_8)$ (Add t_{24}, Suppress t_{22})
			$C(t_{15}, t_9) = C(t_{23}, t_9) \subset t_{18}; C(t_{14}, t_{10}) \subset t_{24}; C(t_{21}, t_{10}) \subset t_5$
			$C(t_{23}, t_{10}) \subset t_{24}; C(t_{19}, t_{13}) \subset t_5; C(t_{16}, t_{14}) \subset C(t_{16}, t_{15}) \subset t_3$
			$C(t_{24}, t_{15}) = t_{23}; C(t_{21}, t_{16}) = t_9; C(t_{23}, t_{16}) = t_{18}$
			$C(t_{21}, t_{18}) = t_9; C(t_{24}, t_{18}) = t_{23}$
t_{13} : 010-	0	1	$C(t_{24}, t_{21})$ (Add t_{25})
~~t_{14} : -100-~~	1	0	$C(t_{24}, t_5)$ (Add t_{26}, Suppress t_{10}, t_{25})
~~t_{15} : 10-1~~	1	0	
~~t_{16} : -00-1~~	1	0	
t_{17} : - -01	0	1	
t_{18} : -0-1	1	0	
t_{19} : - -10	0	1	
t_{20} : -110	1	1	
t_{21} : 011-	1	0	
~~t_{22} : 1-00~~	1	0	
t_{23} : 10- -	1	0	
t_{24} : 1- -0	1	0	
~~t_{25} : -110~~	1	0	
t_{26} : -1-0	1	0	

Table 4.15. *Table for the determination of prime implicants using the consensus method*

and

$$G(A,B,C,D) = C \cdot \overline{D} + \overline{C} \cdot D + \overline{A} \cdot B \cdot \overline{D} \qquad [4.63]$$

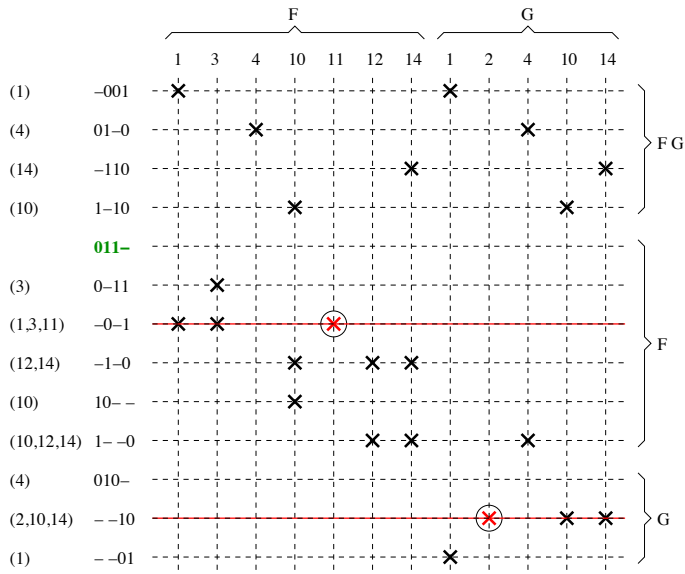


Table 4.16. *Prime implicant chart for the functions F, G and $F \cdot G$*

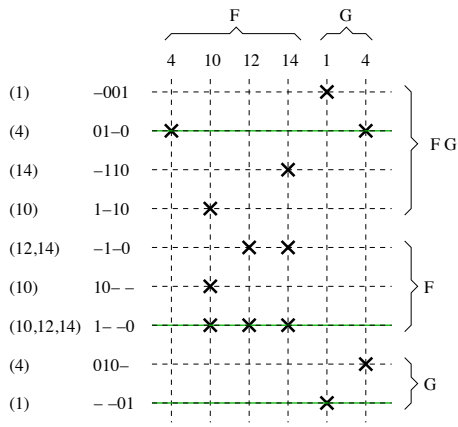


Table 4.17. *Reduced form of the prime implicant chart for the functions F, G and $F \cdot G$*

4.5. Exercises

EXERCISE 4.1.– Karnaugh map with entered variables.

Determine the canonical and minimized sum-of-products forms for each of the logic functions whose Karnaugh map is represented in Figure 4.19.

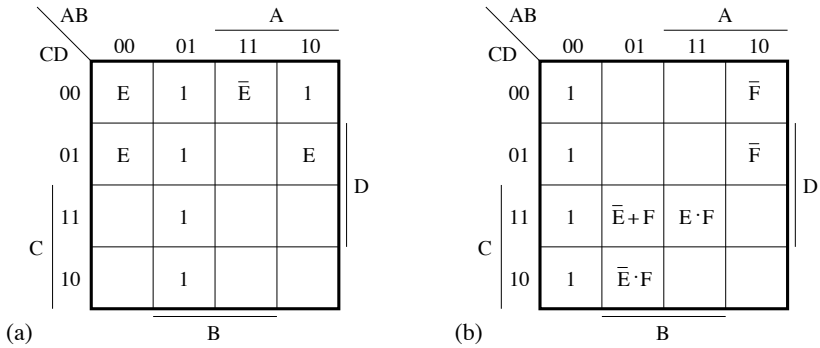


Figure 4.19. *Karnaugh maps*

EXERCISE 4.2.– Karnaugh map with entered variables and incompletely defined logic functions.

Determine the canonical and minimized sum-of-products forms for each of the incompletely defined logic functions whose Karnaugh map is represented in Figure 4.20.

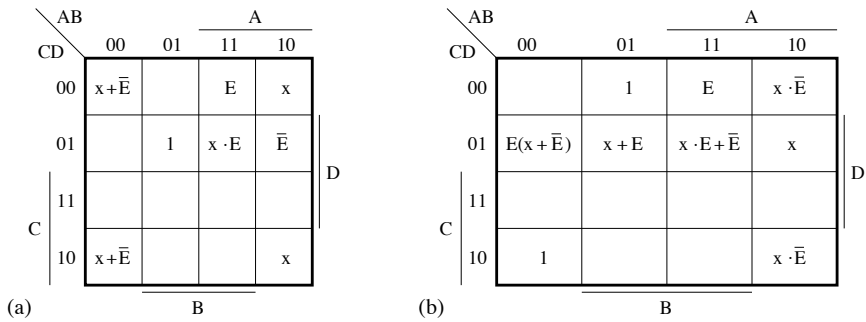


Figure 4.20. *Karnaugh maps*

EXERCISE 4.3.– Karnaugh maps with entered variables and logic functions in the canonical form.

Determine the minimized sum-of-products form for each of the following logic functions:

a)

$$P(A,B,C,D,E) = \sum m(1,6,7,9,13,15,16,17,18,22,23,24,25,28,29) \quad [4.64]$$

b)

$$Q(A,B,C,D,E) = \sum m(2,3,6,9,10,12,13,16,17,22,24,25,26,27,29,31) \quad [4.65]$$

c)

$$R(A,B,C,D,E,F) = \sum m(12,13,14,15,16,17,20,21,24,25,28, 29,30,31,33,35,37,39,49,50,52,53,54,55) \quad [4.66]$$

d)

$$S(A,B,C,D,E,F) = \sum m(2,3,4,5,7,12,13,14,15, 25,29,30,34,35,56,57,58,60,61,63) \quad [4.67]$$

EXERCISE 4.4.– Quine–McCluskey method.

Determine the minimized sum-of-products form for each of the following logic functions using the Quine–McCluskey method:

a)

$$Z_1(A,B,C,D,E) = \sum m(0,2,3,8,10,16,17,18,19,21,24,26) \quad [4.68]$$

b)

$$Z_2(A,B,C,D,E) = \sum m(1,14,16,18,19,22,23,24,30) + \sum x(2,3,5,6,7,17,25,26) \quad [4.69]$$

c)

$$Z_3(A,B,C,D,E,F) = \sum m(10,18,26,40,41,42,48,49,50,52,53,56,57,60,61) \quad [4.70]$$

d)

$$Z_4(A,B,C,D,E,F) = \sum m(0,1,2,3,16,17,18,19,29,44,53,60) + \sum x(12,21,28) \quad [4.71]$$

4.6. Solutions

SOLUTION 4.1.– Karnaugh map with inscribed variables.

a) Based on the Karnaugh map, the logic function Z can be obtained as follows:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot E + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E + \overline{A}\cdot B\cdot\overline{C}\cdot\overline{D} + \\ & \overline{A}\cdot B\cdot\overline{C}\cdot D + \overline{A}\cdot B\cdot C\cdot\overline{D} + \overline{A}\cdot B\cdot C\cdot D + \\ & A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D} + A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E + A\cdot B\cdot\overline{C}\cdot\overline{D}\cdot\overline{E} \end{aligned} \quad [4.72]$$

Using the complement law to bring out only the minterms in the expression of Z, we obtain:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot E + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E + \\ & \overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}\cdot\overline{E} + \overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}\cdot E + \overline{A}\cdot B\cdot\overline{C}\cdot D\cdot\overline{E} + \\ & \overline{A}\cdot B\cdot\overline{C}\cdot D\cdot E + \overline{A}\cdot B\cdot C\cdot\overline{D}\cdot\overline{E} + \overline{A}\cdot B\cdot C\cdot\overline{D}\cdot E + \\ & \overline{A}\cdot B\cdot C\cdot D\cdot\overline{E} + \overline{A}\cdot B\cdot C\cdot D\cdot E + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E} + \\ & A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot E + A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E + A\cdot B\cdot\overline{C}\cdot\overline{D}\cdot\overline{E} \end{aligned} \quad [4.73]$$

Replacing each minterm with the decimal value corresponding to the binary combination of its variables, we have:

$$Z(A,B,C,D,E) = \sum m(1,3,8,9,10,11,12,13,14,15,16,17,19,24) \quad [4.74]$$

The Karnaugh map shown in Figure 4.21(a) can be used to arrive at the following minimized sum-of-products expression:

$$Z(A,B,C,D,E) = \overline{A}\cdot B + \overline{B}\cdot\overline{C}\cdot E + A\cdot\overline{C}\cdot\overline{D}\cdot\overline{E} \quad [4.75]$$

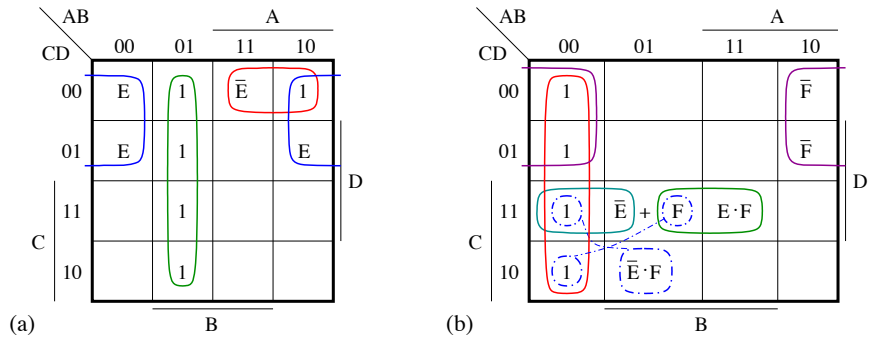


Figure 4.21. *Karnaugh map*

b) Based on the Karnaugh map, the function Z can be written as follows:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D} + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D} + \\ & \overline{A}\cdot\overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot C\cdot D(\overline{E}+F) + \overline{A}\cdot B\cdot C\cdot\overline{D}\cdot\overline{E}\cdot F + \\ & A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{F} + A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot\overline{F} + A\cdot B\cdot C\cdot D\cdot F \end{aligned} \quad [4.76]$$

The logic function Z can then be expressed in the following form:

$$\begin{aligned} Z(A,B,C,D,E,F) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}\cdot\overline{F} + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}\cdot F + \\ & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot E\cdot\overline{F} + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot E\cdot F + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot\overline{E}\cdot\overline{F} + \\ & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot\overline{E}\cdot F + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E\cdot\overline{F} + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E\cdot F + \\ & \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\overline{E}\cdot\overline{F} + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\overline{E}\cdot F + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}\cdot E\cdot\overline{F} + \\ & \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\overline{E}\cdot\overline{F} + \overline{A}\cdot\overline{B}\cdot C\cdot D\cdot\overline{E}\cdot\overline{F} + \overline{A}\cdot\overline{B}\cdot C\cdot D\cdot\overline{E}\cdot F + \\ & \overline{A}\cdot\overline{B}\cdot C\cdot D\cdot E\cdot\overline{F} + \overline{A}\cdot\overline{B}\cdot C\cdot D\cdot E\cdot F + \overline{A}\cdot B\cdot C\cdot\overline{D}\cdot\overline{E}\cdot F + \\ & \overline{A}\cdot B\cdot C\cdot D\cdot\overline{E}\cdot\overline{F} + \overline{A}\cdot B\cdot C\cdot D\cdot\overline{E}\cdot F + \overline{A}\cdot B\cdot C\cdot D\cdot E\cdot F + \\ & A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}\cdot\overline{F} + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot E\cdot\overline{F} + A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot\overline{E}\cdot\overline{F} + \\ & A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E\cdot\overline{F} + A\cdot B\cdot C\cdot D\cdot E\cdot F \end{aligned} \quad [4.77]$$

In decimal form, the canonical equation is given by:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \sum m(0,1,2,3,4,5,6,7,8,9,10,11,12, \\ & 13,14,15,25,28,29,31,32,34,36,38,63) \end{aligned} \quad [4.78]$$

The simplification of Z using the Karnaugh map, as shown in Figure 4.21(b), yields:

$$Z(A,B,C,D,E) = \overline{A}\cdot\overline{B} + \overline{B}\cdot\overline{C}\cdot\overline{F} + \overline{A}\cdot C\cdot D\cdot\overline{E} + \overline{A}\cdot C\cdot\overline{E}\cdot F + B\cdot C\cdot D\cdot E\cdot F \quad [4.79]$$

SOLUTION 4.2.– Karnaugh map with entered variables and incompletely defined logic functions.

a) The expression of the function Z obtained from the Karnaugh map is given by:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}(\mathrm{x}+\overline{E}) + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}(\mathrm{x}+\overline{E}) + \\ & \overline{A}\cdot B\cdot\overline{C}\cdot D + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\mathrm{x} + A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot\overline{E} + \\ & A\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\mathrm{x} + A\cdot B\cdot\overline{C}\cdot\overline{D}\cdot E + A\cdot B\cdot\overline{C}\cdot D\cdot E\cdot\mathrm{x} \end{aligned} \quad [4.80]$$

The function Z can be rewritten as follows:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}(1+\mathrm{x}) + \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot E\cdot\mathrm{x} + \\ & \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\overline{E}(1+\mathrm{x}) + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}\cdot E\cdot\mathrm{x} + \\ & \overline{A}\cdot B\cdot\overline{C}\cdot D\cdot\overline{E} + \overline{A}\cdot B\cdot\overline{C}\cdot D\cdot E + \\ & A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}\cdot\mathrm{x} + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot E\cdot\mathrm{x} + \\ & A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot\overline{E} + A\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\overline{E}\cdot\mathrm{x} + \\ & A\cdot\overline{B}\cdot C\cdot\overline{D}\cdot E\cdot\mathrm{x} + A\cdot B\cdot\overline{C}\cdot\overline{D}\cdot E + A\cdot B\cdot\overline{C}\cdot D\cdot E\cdot\mathrm{x} \end{aligned} \quad [4.81]$$

In the decimal form, the canonical expression of the function Z is given by:

$$Z(A,B,C,D,E) = \sum m(0,4,10,11,18,25) + \sum x(1,5,16,17,20,21,27) \quad [4.82]$$

As it is possible to encircle the term E with either $\mathrm{x}\cdot E$ or with x, the Karnaugh map shown in Figure 4.22(a), where the don't care state is assumed to be set at 1, allows us to write:

$$Z(A,B,C,D,E) = \overline{B}\cdot\overline{D} + \overline{A}\cdot B\cdot\overline{C}\cdot D + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{E} + A\cdot B\cdot\overline{C}\cdot E \quad [4.83]$$

or

$$Z(A,B,C,D,E) = \overline{B}\cdot\overline{D} + \overline{A}\cdot B\cdot\overline{C}\cdot D + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{E} + A\cdot\overline{C}\cdot\overline{D}\cdot E \quad [4.84]$$

b) Based on the Karnaugh map, we have:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E(\mathrm{x}+\overline{E}) + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}+ \\ & \overline{A}\cdot B\cdot\overline{C}\cdot\overline{D} + \overline{A}\cdot B\cdot\overline{C}\cdot D(\mathrm{x}+E)+ \\ & A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}\cdot\mathrm{x} + A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot\mathrm{x}+ \\ & A\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\overline{E}\cdot\mathrm{x} + A\cdot B\cdot\overline{C}\cdot\overline{D}\cdot E + A\cdot B\cdot\overline{C}\cdot D(\mathrm{x}\cdot E+\overline{E}) \end{aligned} \quad [4.85]$$

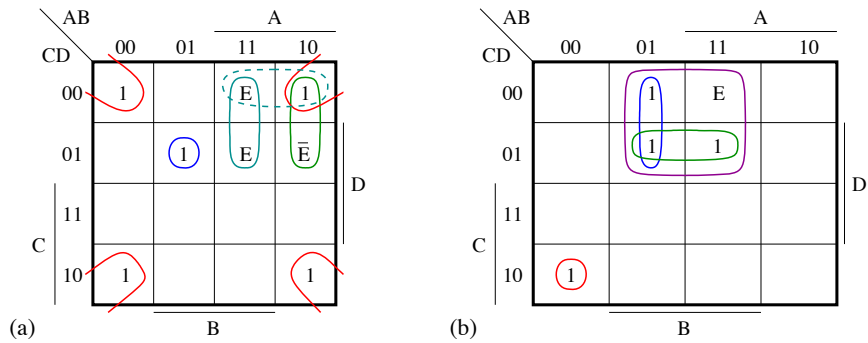


Figure 4.22. *Karnaugh maps*

The expansion of the function Z, taking into account the complement law, translates to:

$$\begin{aligned} Z(A,B,C,D,E) = {} & \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E\cdot\mathrm{x} + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\overline{E}+ \\ & \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}\cdot E + \overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}\cdot\overline{E} + \overline{A}\cdot B\cdot\overline{C}\cdot\overline{D}\cdot E+ \\ & \overline{A}\cdot B\cdot\overline{C}\cdot D\cdot E(1+\mathrm{x}) + \overline{A}\cdot B\cdot\overline{C}\cdot D\cdot\overline{E}\cdot\mathrm{x} + A\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}\cdot\overline{E}\cdot\mathrm{x}+ \\ & A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot\overline{E}\cdot\mathrm{x} + A\cdot\overline{B}\cdot\overline{C}\cdot D\cdot E\cdot\mathrm{x} + A\cdot\overline{B}\cdot C\cdot\overline{D}\cdot\overline{E}\cdot\mathrm{x}+ \\ & A\cdot B\cdot\overline{C}\cdot\overline{D}\cdot E + A\cdot B\cdot\overline{C}\cdot D\cdot E\cdot\mathrm{x} + A\cdot B\cdot\overline{C}\cdot D\cdot\overline{E} \end{aligned} \quad [4.86]$$

The function Z can equivalently be defined by:

$$Z(A,B,C,D,E) = \sum m(4,5,8,9,11,25,26) + \sum x(3,10,16,18,19,20,27) \quad [4.87]$$

Using the Karnaugh map shown in Figure 4.22(b), where $x_3 = x_{16} = x_{18} = x_{19} = x_{20} = 0$ and $x_{10} = x_{27} = 1$, we obtain:

$$Z(A,B,C,D,E) = B\cdot\overline{C}\cdot E + \overline{A}\cdot B\cdot\overline{C} + B\cdot\overline{C}\cdot D + \overline{A}\cdot\overline{B}\cdot C\cdot\overline{D} \quad [4.88]$$

SOLUTION 4.3.– Karnaugh maps with entered variables and logic functions in the canonical form.

a) Based on the Karnaugh map shown in Figure 4.23(a), the minimized sum-of-products form for the logic function P can be written as:

$$P(A,B,C,D,E) = A\cdot B\cdot \overline{D} + \overline{B}\cdot C\cdot D + \overline{C}\cdot \overline{D}\cdot E + A\cdot \overline{B}\cdot \overline{C}\cdot \overline{E} + \overline{A}\cdot B\cdot C\cdot E \quad [4.89]$$

b) With reference to the Karnaugh map shown in Figure 4.23(b), the following minimized expression can be obtained:

$$Q(A,B,C,D,E) = A\cdot B\cdot E + A\cdot \overline{C}\cdot \overline{D} + B\cdot \overline{D}\cdot E + \overline{A}\cdot B\cdot C\cdot \overline{D} + \overline{A}\cdot \overline{B}\cdot \overline{C}\cdot D + B\cdot \overline{C}\cdot D\cdot \overline{E} + \overline{B}\cdot C\cdot D\cdot \overline{E} \quad [4.90]$$

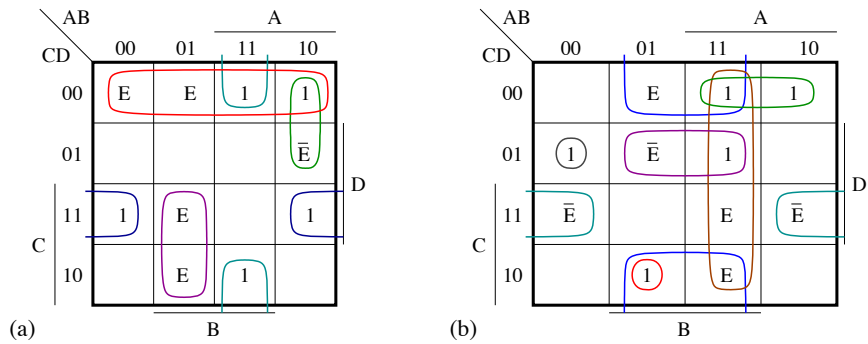


Figure 4.23. *Karnaugh maps*

c) Figure 4.24(a) depicts the Karnaugh map for the function R. As the term $\overline{E}\cdot F$ can be covered by either F or by $\overline{E}$, we have the following two solutions:

$$R(A,B,C,D,E,F) = \overline{A}\cdot C\cdot D + \overline{A}\cdot B\cdot \overline{E} + A\cdot B\cdot \overline{C}\cdot D + A\cdot \overline{B}\cdot \overline{C}\cdot F + A\cdot B\cdot \overline{C}\cdot E\cdot \overline{F} + A\cdot \overline{C}\cdot \overline{E}\cdot F \quad [4.91]$$

or

$$R(A,B,C,D,E,F) = \overline{A}\cdot C\cdot D + \overline{A}\cdot B\cdot \overline{E} + A\cdot B\cdot \overline{C}\cdot D + A\cdot \overline{B}\cdot \overline{C}\cdot F + A\cdot B\cdot \overline{C}\cdot E\cdot \overline{F} + B\cdot \overline{C}\cdot \overline{E}\cdot F \quad [4.92]$$

d) Because $\overline{E} + F = \overline{E \cdot \overline{F}}$, based on the Karnaugh map shown in Figure 4.24(b) we have:

$$\begin{aligned} S(A,B,C,D,E,F) = {} & A \cdot B \cdot C \cdot \overline{E} + \overline{A} \cdot \overline{B} \cdot D \cdot \overline{E} + \overline{A} \cdot \overline{B} \cdot D \cdot F + \\ & B \cdot C \cdot \overline{E} \cdot F + \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot E + A \cdot B \cdot C \cdot D \cdot F + \\ & A \cdot B \cdot C \cdot \overline{D} \cdot \overline{F} + \overline{A} \cdot C \cdot D \cdot E \cdot \overline{F} \end{aligned} \quad [4.93]$$

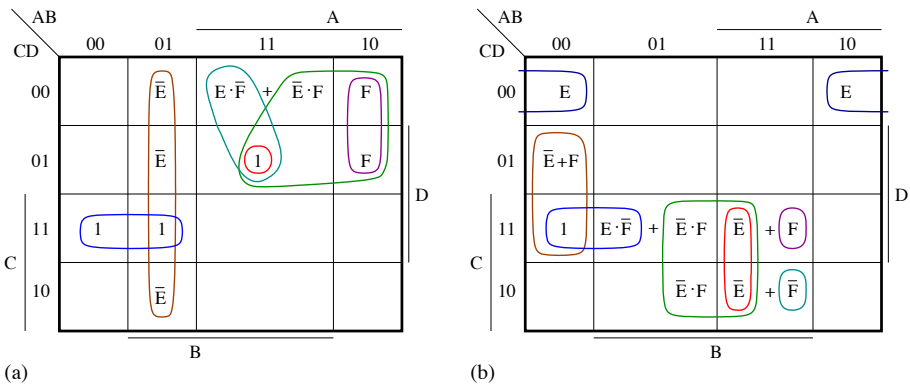


Figure 4.24. *Karnaugh maps*

SOLUTION 4.4.– The Quine–McCluskey method

a)

$$Z_1(A,B,C,D,E) = \sum m(0,2,3,8,10,16,17,18,19,21,24,26) \quad [4.94]$$

Table 4.18 gives the following prime implicants:

$$A \cdot \overline{B} \cdot \overline{D} \cdot E, \quad \overline{B} \cdot \overline{C} \cdot D, \quad A \cdot \overline{B} \cdot \overline{C} \quad \text{and} \quad \overline{C} \cdot \overline{E}$$

Table 4.19 gives the prime implicant chart for the function Z_1. As all the terms of Z_1 are covered by the essential prime implicants, that is - -00 $(\overline{C} \cdot \overline{E})$, -001- $(\overline{B} \cdot \overline{C} \cdot D)$ and 10-01 $(A \cdot \overline{B} \cdot \overline{D} \cdot E)$, the minimized sum-of-products forms for Z_1 can be written as:

$$Z_1(A,B,C,D,E) = \overline{C} \cdot \overline{E} + \overline{B} \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot \overline{D} \cdot E \quad [4.95]$$

b)

$$Z_2(A,B,C,D,E) = \sum m(1,14,16,18,19,22,23,24,30) + \sum x(2,3,5,6,7,17,25,26) \quad [4.96]$$

	Column 1	Column 2	Column 3	Column 4
Group 0	m_0 : 00000 ✓	t_1 : 000-0 ✓ t_2 : 0-000 ✓ t_3 : -0000 ✓	t_{19} : 0-0-0 ✓ t_{20} : -00-0 ✓ t_{21} : - -000 ✓	t_{27} : - -0-0 ⋆
Group 1	m_2 : 00010 ✓ m_8 : 01000 ✓ m_{16} : 10000 ✓	t_4 : 0001- ✓ t_5 : 0-010 ✓ t_6 : 010-0 ✓ t_7 : 1000- ✓ t_8 : -0010 ✓ t_9 : 100-0 ✓ t_{10} : -1000 ✓ t_{11} : 1-000 ✓	t_{22} : -001- ⋆ t_{23} : 100- - ⋆ t_{24} : - -010 ✓ t_{25} : -10-0 ✓ t_{26} : 1-0-0 ✓	
Group 2	m_3 : 00011 ✓ m_{10} : 01010 ✓ m_{17} : 10001 ✓ m_{18} : 10010 ✓ m_{24} : 11000 ✓	t_{12} : -0011 ✓ t_{13} : 100-1 ✓ t_{14} : 1001- ✓ t_{15} : 10-01 ⋆ t_{16} : -1010 ✓ t_{17} : 1-010 ✓ t_{18} : 110-0 ✓		
Group 3	m_{19} : 10011 ✓ m_{21} : 10101 ✓ m_{26} : 11010 ✓			

Table 4.18. *Table for the determination of the prime implicants of Z_1*

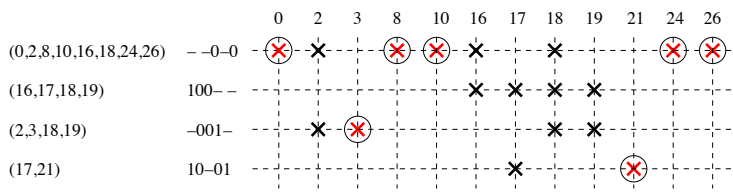


Table 4.19. *Prime implicant chart for Z_1*

	Column 1	Column 2	Column 3	Column 4
Group 1	m_1 : 00001 ✓ m_2 : 00010 ✓ m_{16} : 10000 ✓	t_1 : 000-1 ✓ t_2 : 0001- ✓ t_3 : 00-01 ✓ t_4 : 00-10 ✓ t_5 : -0001 ✓ t_6 : 1000- ✓ t_7 : -0010 ✓ t_8 : 100-0 ✓ t_9 : 1-000 ✓	t_{29} : 00- - 1 ⋆ t_{30} : 00-1- ✓ t_{31} : -00-1 ⋆ t_{32} : -001- ✓ t_{33} : 100- - ⋆ t_{34} : -0-10 ✓ t_{35} : 1-00- ⋆ t_{36} : 1-0-0 ⋆	t_{43} : -0-1- ⋆
Group 2	m_3 : 00011 ✓ m_5 : 00101 ✓ m_6 : 00110 ✓ m_{17} : 10001 ✓ m_{18} : 10010 ✓ m_{24} : 11000 ✓	t_{10} : 00-11 ✓ t_{11} : 001-1 ⋆ t_{12} : 0011- ✓ t_{13} : 0-110 ✓ t_{14} : -0011 ✓ t_{15} : 100-1 ⋆ t_{16} : 1001- ✓ t_{17} : -0110 ✓ t_{18} : 10-10 ✓ t_{19} : 1-001 ⋆ t_{20} : 1100- ⋆ t_{21} : 1-010 ✓ t_{22} : 110-0 ⋆	t_{37} : -0-11 ✓ t_{38} : -011- ✓ t_{39} : -0-11 ✓ t_{40} : 10-1- ✓ t_{41} : - -110 ⋆ t_{42} : 1- -10 ⋆	
Group 3	m_7 : 00111 ✓ m_{14} : 01110 ✓ m_{19} : 10011 ✓ m_{22} : 10110 ✓ m_{25} : 11001 ✓ m_{26} : 11010 ✓	t_{23} : -0111 ✓ t_{24} : 10-11 ✓ t_{25} : 1011- ✓ t_{26} : -1110 ✓ t_{27} : 1-110 ✓ t_{28} : 11-10 ✓		
Group 4	m_{23} : 10111 ✓ m_{30} : 11110 ✓			

Table 4.20. *Table for the determination of the prime implicants of Z_2*

According to Table 4.20, the logic function Z_2 has 13 prime implicants. As shown in the prime implicant chart in Table 4.21(a), the essential prime implicants are -0-1- $(\overline{B} \cdot D)$ and - -110 $(C \cdot D \cdot \overline{E})$. The two implicants 1-001 and 001-1 cover no minterm of Z_2 as they have been obtained by combining only the minterms (25 and 17, and 7 and 5) corresponding to don't care states.

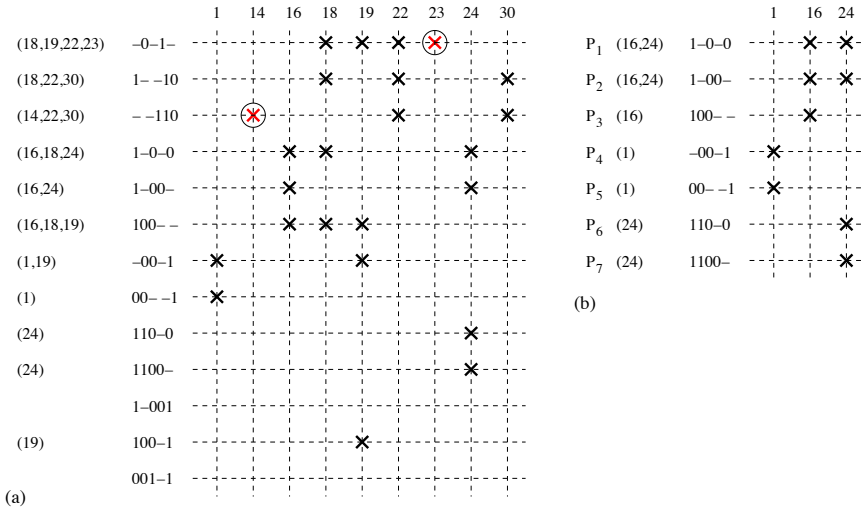


Table 4.21. *Prime implicant chart for Z_2*

Table 4.21(b) gives the reduced form of the prime implicant chart. Using Petrick's method to complete the choice of prime implicants, we have:

$$P = (P_4 + P_5)(P_1 + P_2 + P_3)(P_1 + P_2 + P_6 + P_7) \qquad [4.97]$$

The expansion of P, taking into account the logic identity $1 + X = 1$, yields:

$$P = P_1P_4 + P_1P_5 + P_2P_4 + P_2P_5 + P_3P_6(P_4 + P_5) + P_3P_7(P_4 + P_5) \qquad [4.98]$$

Retaining only the products with the minimum terms, that is P_1P_4, P_1P_5, P_2P_4 and P_2P_5, where P_1, P_2, P_4 and P_5 represent, respectively, the terms 1-0-0 $(A \cdot \overline{C} \cdot \overline{E})$, 1-00- $(A \cdot \overline{C} \cdot \overline{D})$, -00-1 $(\overline{B} \cdot \overline{C} \cdot E)$ and 00- -1 $(\overline{A} \cdot \overline{B} \cdot E)$, we obtain four possible solutions. Thus:

$$Z_2(A, B, C, D, E) = \overline{B} \cdot D + C \cdot D \cdot \overline{E} + A \cdot \overline{C} \cdot \overline{E} + \overline{B} \cdot \overline{C} \cdot E \qquad [4.99]$$

$$Z_2(A, B, C, D, E) = \overline{B} \cdot D + C \cdot D \cdot \overline{E} + A \cdot \overline{C} \cdot \overline{E} + \overline{A} \cdot \overline{B} \cdot E \qquad [4.100]$$

$$Z_2(A, B, C, D, E) = \overline{B} \cdot D + C \cdot D \cdot \overline{E} + A \cdot \overline{C} \cdot \overline{D} + \overline{B} \cdot \overline{C} \cdot E \qquad [4.101]$$

and

$$Z_2(A, B, C, D, E) = \overline{B} \cdot D + C \cdot D \cdot \overline{E} + A \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot E \qquad [4.102]$$

c)

$$Z_3(A,B,C,D,E,F) = \sum m(10,18,26,40,41,42,48,49,50,52,53, 56,57,60,61) \quad [4.103]$$

Referring to Table 4.22, the function Z_3 has nine prime implicants. Table 4.23 corresponds to the prime implicant chart for Z_3, where only a single essential prime implicant can be identified, that is 11- -0- $(A \cdot B \cdot \overline{E})$. Table 4.24 gives the reduced form of the prime implicant chart obtained by considering only the non-essential prime implicants. Using Petrick's methods to complete the choice of prime implicants leads to a logic equation that can be written as follows:

$$P = (P_6 + P_8)(P_4 + P_7)(P_7 + P_8)(P_1 + P_5)(P_1 + P_2) \times (P_5 + P_6)(P_3 + P_4) \quad [4.104]$$

	Column 1	Column 2	Column 3	Column 4
Group 2	m_{10} : 001010 ✓ m_{18} : 010010 ✓ m_{40} : 101000 ✓ m_{48} : 110000 ✓	t_1 : 0-1010 ⋆ t_2 : 01-010 ⋆ t_3 : 10100- ✓ t_4 : -01010 ⋆ t_5 : 1010-0 ⋆ t_6 : 11000- ✓ t_7 : -10010 ⋆ t_8 : 1100-0 ⋆ t_9 : 110-00 ✓ t_{10} : 1-1000 ✓ t_{11} : 11-000 ✓	t_{22} : 110-0- ✓ t_{23} : 1-100- ⋆ t_{24} : 11-00- ✓ t_{25} : 11- -00 ✓	t_{29} : 11- -0- ⋆
Group 3	m_{26} : 011010 ✓ m_{41} : 101001 ✓ m_{42} : 101010 ✓ m_{49} : 110001 ✓ m_{50} : 110010 ✓ m_{52} : 110100 ✓ m_{56} : 111000 ✓	t_{12} : 110-01 ✓ t_{13} : 11010- ✓ t_{14} : 1-1001 ⋆ t_{15} : 11-001 ✓ t_{16} : 11100- ✓ t_{17} : 11-100 ✓ t_{18} : 111-00 ✓	t_{26} : 11- -01 ✓ t_{27} : 11-10- ✓ t_{28} : 111-0- ✓	
Group 4	m_{53} : 110101 ✓ m_{57} : 111001 ✓ m_{60} : 111100 ✓	t_{19} : 11-101 ✓ t_{20} : 111-01 ✓ t_{21} : 11110- ✓		
Group 5	m_{61} : 111101 ✓			

Table 4.22. *Table for the determination of the prime implicants of Z_3*

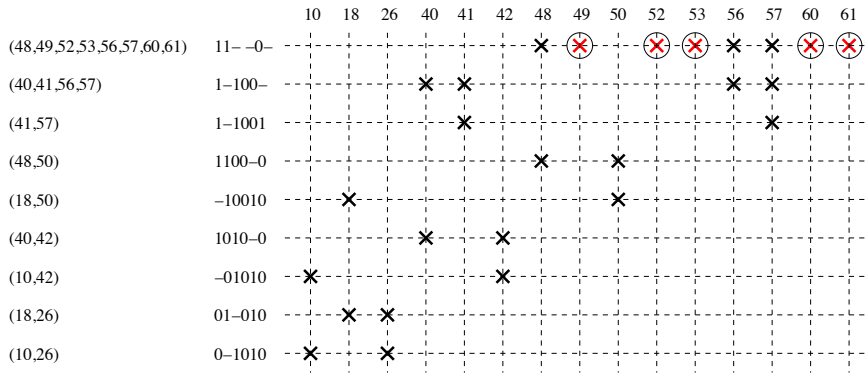


Table 4.23. *Prime implicant chart for Z_3*

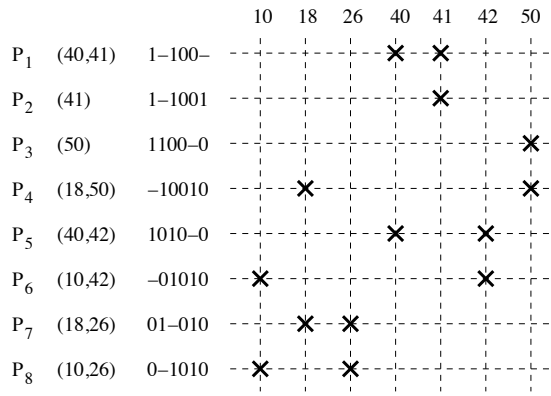


Table 4.24. *Reduced prime implicant chart for Z_3*

Expanding the expression for P and taking into account the logic identity $1+X=1$, we arrive at:

$$\begin{aligned} P = {} & P_1P_4P_5P_8 + P_1P_4P_6P_8 + P_1P_4P_6P_7 + P_1P_3P_6P_7 + \\ & P_2P_4P_5P_8 + (P_3P_6P_7 + P_4P_6P_7)(P_1P_5 + P_2P_5) + \\ & P_3P_7P_8(P_1P_5 + P_1P_6 + P_2P_5) \end{aligned} \qquad [4.105]$$

As the term associated with P_2 has one variable more than that corresponding to P_1, we select only the first four products, that is $P_1P_4P_5P_8$, $P_1P_4P_6P_8$, $P_1P_4P_6P_7$ and $P_1P_3P_6P_7$, where P_1, P_3, P_4, P_5, P_6, P_7 and P_8 represent, respectively, the terms

1-100- ($A \cdot C \cdot \overline{D} \cdot \overline{E}$), 1100-0 ($A \cdot B \cdot \overline{C} \cdot \overline{D} \cdot \overline{F}$), -10010 ($B \cdot \overline{C} \cdot \overline{D} \cdot E \cdot \overline{F}$), 1010-0 ($A \cdot \overline{B} \cdot C \cdot \overline{D} \cdot \overline{F}$), -01010 ($\overline{B} \cdot C \cdot \overline{D} \cdot E \cdot \overline{F}$), 01-010 ($\overline{A} \cdot B \cdot \overline{D} \cdot E \cdot \overline{F}$) and 0-1010 ($\overline{A} \cdot C \cdot \overline{D} \cdot E \cdot \overline{F}$). Finally, the four minimized sum-of-products forms for the function Z_3 are given by:

$$Z_3 = A \cdot B \cdot \overline{E} + A \cdot C \cdot \overline{D} \cdot \overline{E} + B \cdot \overline{C} \cdot \overline{D} \cdot E \cdot \overline{F} + A \cdot \overline{B} \cdot C \cdot \overline{D} \cdot \overline{F} + \overline{A} \cdot C \cdot \overline{D} \cdot E \cdot \overline{F} \quad [4.106]$$

$$Z_3 = A \cdot B \cdot \overline{E} + A \cdot C \cdot \overline{D} \cdot \overline{E} + B \cdot \overline{C} \cdot \overline{D} \cdot E \cdot \overline{F} + \overline{B} \cdot C \cdot \overline{D} \cdot E \cdot \overline{F} + \overline{A} \cdot C \cdot \overline{D} \cdot E \cdot \overline{F} \quad [4.107]$$

$$Z_3 = A \cdot B \cdot \overline{E} + A \cdot C \cdot \overline{D} \cdot \overline{E} + B \cdot \overline{C} \cdot \overline{D} \cdot E \cdot \overline{F} + \overline{B} \cdot C \cdot \overline{D} \cdot E \cdot \overline{F} + \overline{A} \cdot B \cdot \overline{D} \cdot E \cdot \overline{F} \quad [4.108]$$

and

$$Z_3 = A \cdot B \cdot \overline{E} + A \cdot C \cdot \overline{D} \cdot \overline{E} + A \cdot B \cdot \overline{C} \cdot \overline{D} \cdot \overline{F} + \overline{B} \cdot C \cdot \overline{D} \cdot E \cdot \overline{F} + \overline{A} \cdot B \cdot \overline{D} \cdot E \cdot \overline{F} \quad [4.109]$$

d)

$$Z_4(A, B, C, D, E, F) = \sum m(0, 1, 2, 3, 16, 17, 18, 19, 29, 44, 53, 60) + \sum x(12, 21, 28) \quad [4.110]$$

According to Table 4.25, the logic function Z_4 has six prime implicants. On going through the prime implicant chart, as given in Table 4.26, we can observe that the three essential prime implicants, 0-00- - ($\overline{A} \cdot \overline{C} \cdot \overline{D}$), - -1100 ($C \cdot D \cdot \overline{E} \cdot \overline{F}$) and -10101 ($B \cdot \overline{C} \cdot D \cdot \overline{E} \cdot F$), cover all the minterms except for the minterm 29. As the minterm 29 can be covered by either 01110- ($\overline{A} \cdot B \cdot C \cdot D \cdot \overline{E}$), or by 01-101 ($\overline{A} \cdot B \cdot D \cdot \overline{E} \cdot F$), the function Z_4 has two minimized forms. Thus:

$$Z_4 = B \cdot \overline{C} \cdot \overline{D} + C \cdot D \cdot \overline{E} \cdot \overline{F} + B \cdot \overline{C} \cdot D \cdot \overline{E} \cdot F + \overline{A} \cdot B \cdot C \cdot D \cdot \overline{E} \quad [4.111]$$

and

$$Z_4 = B \cdot \overline{C} \cdot \overline{D} + C \cdot D \cdot \overline{E} \cdot \overline{F} + B \cdot \overline{C} \cdot D \cdot \overline{E} \cdot F + \overline{A} \cdot B \cdot D \cdot \overline{E} \cdot F \quad [4.112]$$

	Column 1	Column 2	Column 3	Column 4
Group 0	m_0 : 000000 ✓	t_1 : 00000- ✓ t_2 : 0000-0 ✓ t_3 : 0-0000 ✓	t_{21} : 0000- - ✓ t_{22} : 0-000- ✓ t_{23} : 0-00-0 ✓	t_{28} : 0-00- - ⋆
Group 1	m_1 : 000001 ✓ m_2 : 000010 ✓ m_{16} : 010000 ✓	t_4 : 0000-1 ✓ t_5 : 00001- ✓ t_6 : 0-0001 ✓ t_7 : 01000- ✓ t_8 : 0-0010 ✓ t_9 : 0100-0 ✓	t_{24} : 0-00-1 ✓ t_{25} : 0-001- ✓ t_{26} : 0100- - ✓	
Group 2	m_3 : 000011 ✓ m_{12} : 001100 ✓ m_{17} : 010001 ✓ m_{18} : 010010 ✓	t_{10} : 0-0011 ✓ t_{11} : 0100-1 ✓ t_{12} : 01001- ✓ t_{13} : 010-01 ⋆ t_{14} : 0-1100 ✓ t_{15} : -01100 ✓	t_{27} : - -1100 ⋆	
Group 3	m_{19} : 010011 ✓ m_{21} : 010101 ✓ m_{28} : 011100 ✓ m_{44} : 101100 ✓ m_{44} : 101100 ✓	t_{16} : 01-101 ⋆ t_{17} : 01110- ⋆ t_{18} : -10101 ⋆ t_{19} : -11100 ✓ t_{20} : 1-1100 ✓		
Group 4	m_{29} : 011101 ✓ m_{53} : 110101 ✓ m_{60} : 111100 ✓			

Table 4.25. *Table for the determination of the prime implicants of* Z_4

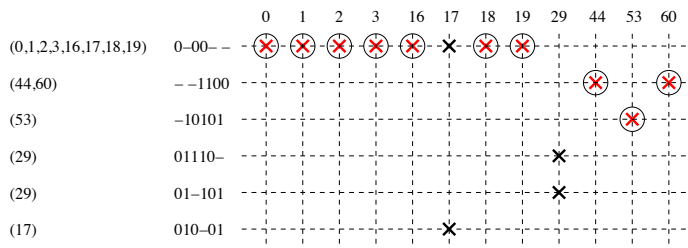


Table 4.26. *Prime implicant chart for* Z_4

Bibliography

[BRO 08] BROWN S., VRANESIC Z., *Fundamentals of Digital Logic with VHDL Design*, 3rd ed., McGraw-Hill Education, New York City, NY, 2008.

[CLE 00] CLEMENTS A., *The Principles of Computer Hardware*, 3rd ed., Oxford University Press, Oxford, UK, 2000.

[COM 95] COMER D.J., *Digital Logic and State Machine Design*, 3rd ed., Oxford University Press, New York City, NY, 1995.

[DUE 01] DUECK R.K., *Digital Design with CPLD Applications and VHDL*, Delmar Thomson Learning, Albany, NY, 2001.

[GIV 03] GIVONE D., *Digital Principles and Design*, McGraw-Hill, New York City, NY, 2003.

[HAY 93] HAYES J.P., *Introduction to Digital Logic Design*, Addison-Wesley Publishing Company, Boston, MA, 1993.

[HAY 98] HAYES J.P., *Computer Architecture and Organization*, McGraw-Hill, New York City, NY, 1998.

[KAT 05] KATZ R.H., Borrielo G., *Contemporary Logic Design*, 2nd ed., Prentice Hall, Upper Saddle River, NJ, 2005.

[MAN 01] MANO M.M., *Digital Design*, 3rd ed., Prentice Hall, Upper Saddle River, NJ, 2001.

[MAR 10] MARCOVITZ A.B., *Introduction to Logic Design*, 3rd ed., McGraw-Hill Education, New York City, NY, 2010.

[NDJ 11] NDJOUNTCHE T., *CMOS Analog Integrated Circuits: High-Speed and Power-efficient Design*, CRC Press, Boca Raton, FL, 2011.

[ROT 04] ROTH JR. C.H., *Fundamental of Logic Design*, 5th ed., Brooks/Cole – Thomson Learning, Belmont, CA, 2004.

[SAN 02] SANDIGE R.S., *Digital Design Essentials*, Prentice Hall, Upper Saddle River, NJ, 2002.

[TIN 00] TINDER R.F., *Engineering Digital Design*, Academic Press, San Diego, CA, 2000.

Digital Electronics 1: *Combinational Logic Circuits*, First Edition. Tertulien Ndjountche.

© ISTE Ltd 2016. Published by ISTE Ltd and John Wiley & Sons, Inc.

[TOC 03] TOCCI R.J., AMBROSIO F.J., *Microprocessors and Microcomputers*, 6th ed., Prentice Hall, Upper Saddle River, NJ, 2003.

[WAK 00] WAKERLY J.F., *Digital Design Principles and Practices*, 3rd ed., Prentice Hall, Upper Saddle River, NJ, 2000.

[WIL 98] WILKINSON B., *The Essence of Digital Design*, Prentice Hall Europe, Hemel Hempstead, UK, 1998.

[YAR 97] YARBROUGH J.M., *Digital Logic – Applications and Design*, West Publishing Company, St. Paul, MN, 1997.

Index

A

addition, 16
alphanumeric, 28
AND, 49, 51
ANSI, 31
arithmetic operation, 16
  addition, 16
  division, 19
  multiplication, 18
  subtraction, 17
ASCII, 31, 33

B

barrel shifter, 160, 163
base, 1, 6
BCD, 7
binary, 2, 7, 16, 18
Boole, 49, 57, 59, 64
Boolean algebra, 57, 59, 64
buffer, 54
byte, 3

C

canonical, 221
  form, 56
chart, 225
circuit
  electric, 49
  logic, 59
  multi-level, 76
  two-level, 76
code
  ASCII, 31
  BCD, 149
  binary, 2, 7, 8
  block, 33
  correcting, 33
  cyclic, 34
  excess-3, 149
  gray, 28, 143, 144
  natural binary, 2, 10
  p-out-of-n, 29
  reflected binary, 28
  universal, 31
  XS-3, 149
consensus, 223, 224, 237

D

data, 28
decimal, 18
decoder, 121, 126, 129
DeMorgan, 57, 78, 80
demultiplexer, 121, 126
digit, 1
dividend, 19
division, 2, 5, 19
divisor, 19
don't care term, 69, 73
duad, 66

Digital Electronics 1: *Combinational Logic Circuits*, First Edition. Tertulien Ndjountche.
© ISTE Ltd 2016. Published by ISTE Ltd and John Wiley & Sons, Inc.

E, F

EBCDIC, 31
electric circuit, 49, 50
encoder, 130, 133
entered variable, 208
excess-E, 12
factorization, 74
fixed-point, 20
floating-point, 22, 23
fractional, 13, 15, 21

G, H, I

generator polynomial, 34
gray code, 28
Hamming distance, 33
hazard
    dynamic, 92
    static, 90
hexadecimal, 5, 7
IEEE-754, 22–24, 26
implicant, 204, 221, 223, 225, 227, 230, 231, 237
    essential, 204
    prime, 204

K, L

Karnaugh, 65, 67, 83, 205
Karnaugh map, 65–73, 203, 205, 208, 216
logic function, 53–55, 205
    incompletely defined, 235
    multiple outputs, 73, 235, 237
logic gate, 49–53
    AND, 49, 51
    NAND, 53
    NOR, 53
    NOT, 49, 51
    OR, 49, 52
    universal, 53
    XNOR, 53
    XOR, 50, 52
LSB, 2

M

maxterm, 56, 65, 208
microprocessor, 2
minterm, 55, 56, 65, 69, 203, 204, 208, 215, 217, 220, 225, 227, 228, 231, 237
minuend, 17
MSB, 2
multi-level, 76
multiplexer, 115, 120, 127, 163
multiplicand, 19
multiplication, 18, 21
multiplier, 19

N

NAND, 53, 79
negative logic, 3
NOR, 53, 80
NOT, 49, 51
number
    binary, 16
    fractional, 13, 15
    integer, 8, 13
    real, 20, 28
    signed, 8, 12, 13
    unsigned, 13
number system, 1
    binary, 2
    binary-coded decimal, 7
    decimal, 1
    hexadecimal, 5
    octal, 4

O, P

octad, 66
octal, 4, 7
operand, 17
OR, 49, 52
parity bit, 31, 155
Petrick, 227, 233
polynomial generator, 35
positive logic, 3
prime implicant, 233
prime implicant chart, 225, 237
    cyclic, 226, 227
    graphic reduction, 225
    Petrick's method, 227
priority encoder, 136, 139, 143
product of sums, 55, 205, 208
propagation delay, 90–92

Q, R

quad, 66
Quine–McCluskey, 203, 221, 235, 236
radix, 1
real, 20
Reed-Muller, 83
representation
    BCD, 7
    binary, 10, 14
    decimal, 1
    excess-E, 12
    fixed-point, 20
    floating-point, 22
    hexadecimal, 14
    IEEE-754, 23, 24, 26
    in a base B, 6
    octal, 14
    sign-magnitude, 9, 20
    two's complement, 10, 11, 21

S

Shannon, 60
sign-magnitude, 9, 20
simplification, 73, 235
    algebraic method, 59
    semi-graphical method, 65
    systematic method, 203
subtraction, 17
subtrahend, 17
sum of products, 55, 82, 205, 208, 230, 248
supply voltage, 89

T, W, X

three-state buffer, 54
timing diagram, 89, 90
transcoder, 143
truth table, 55, 60, 61, 90, 115, 120, 121, 123, 128, 131, 134, 136, 137, 143, 156, 163, 209
two's complement, 10, 11, 21
word, 2
XNOR, 53, 57, 61
XOR, 50, 52, 57, 61, 82
XOR gate, 220

Other titles from

in

Electronics Engineering

2015

DURAFFOURG Laurent, ARCAMONE Julien
Nanoelectromechanical Systems

2014

APPRIOU Alain
Uncertainty Theories and Multisensor Data Fusion

CONSONNI Vincent, FEUILLET Guy
Wide Band Gap Semiconductor Nanowires 1: Low-Dimensionality Effects and Growth
Wide Band Gap Semiconductor Nanowires 2: Heterostructures and Optoelectronic Devices

GAUTIER Jean-Luc
Design of Microwave Active Devices

LACAZE Pierre Camille, LACROIX Jean-Christophe
Non-volatile Memories

TEMPLIER François
OLED Microdisplays: Technology and Applications

THOMAS Jean-Hugh, YAAKOUBI Nourdin
New Sensors and Processing Chain

2013

COSTA François, GAUTIER Cyrille, LABOURE Eric, REVOL Bertrand
Electromagnetic Compatibility in Power Electronics

KORDON Fabrice, HUGUES Jérôme, CANALS Agusti, DOHET Alain
Embedded Systems: Analysis and Modeling with SysML, UML and AADL

LE TIEC Yannick
Chemistry in Microelectronics

2012

BECHERRAWY Tamer
Electromagnetism: Maxwell Equations, Wave Propagation and Emission

LALAUZE René
Chemical Sensors and Biosensors

LE MENN Marc
Instrumentation and Metrology in Oceanography

SAGUET Pierre
Numerical Analysis in Electromagnetics: The TLM Method

2011

ALGANI Catherine, RUMELHARD Christian, BILLABERT Anne-Laure
Microwaves Photonic Links: Components and Circuits

BAUDRANT Annie
Silicon Technologies: Ion Implantation and Thermal Treatment

DEFAY Emmanuel
Integration of Ferroelectric and Piezoelectric Thin Films: Concepts ans Applications for Microsystems

DEFAY Emmanuel
Ferroelectric Dielectrics Integrated on Silicon

BESNIER Philippe, DÉMOULIN Bernard
Electromagnetic Reverberation Chambers

LANDIS Stefan
Nano-lithography

2010

LANDIS Stefan
Lithography

PIETTE Bernard
VHF / UHF Filters and Multicouplers

2009

DE SALVO Barbara
Silicon Non-volatile Memories / Paths of Innovation

DECOSTER Didier, HARARI Joseph
Optoelectronic Sensors

FABRY Pierre, FOULETIER Jacques
Chemical and Biological Microsensors / Applications in Fluid Media

GAUTIER Jacques
Physics and Operation of Silicon Devices in Integrated Circuits

MOLITON André
Solid-State Physics for Electronics

PERRET Robert
Power Electronics Semiconductor Devices

SAGUET Pierre
Passive RF Integrated Circuits

2008

CHARRUAU Stéphane
Electromagnetism and Interconnections

2007

RIPKA Pavel, TIPEK Alois
Modern Sensors Handbook

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.

The omnipresence of electronic devices in our everyday lives has been accompanied by the downscaling of chip feature sizes and the ever increasing complexity of digital circuits.

This book is devoted to the analysis and design of digital circuits, where the signal can assume only two possible logic levels. It deals with the basic principles and concepts of digital electronics. It addresses all aspects of combinational logic and provides a detailed understanding of logic gates that are the basic components in the implementation of circuits used to perform functions and operations of Boolean algebra. Combinational logic circuits are characterized by outputs that depend only on the actual input values.

Efficient techniques to derive logic equations are proposed together with methods of analysis and synthesis of combinational logic circuits. Each chapter is well structured and is supplemented by a selection of solved exercises covering logic design practices.

Tertulien Ndjountche received a PhD degree in electrical engineering from Erlangen-Nuremberg University in Germany. He has worked as a professor and researcher at universities in Germany and Canada. He has published numerous technical papers and books in his fields of interest.

ISTE
www.iste.co.uk
WILEY
9 781848 219847